INTRUDETECTOR: A SOFTWARE PLATFORM FOR TESTING NETWORK INTRUSION DETECTION ALGORITHMS

A Thesis
Submitted to the Faculty of Graduate Studies and Research
In Partial Fulfillment of the Requirements
for the Degree of
Master of Science
in
Computer Science
University of Regina

By
Tao Wan
Regina, Saskatchewan
December 2000

© Copyright 2000: Tao Wan
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-60258-3
Abstract

Computer intrusions are prevalent over the Internet, and are becoming a major threat to our society. Many technologies, such as authentication, firewall, among others, have been proposed and utilized to protect our information infrastructure, but none of them suffices. Recently, Intrusion Detection Systems (IDSs) have been receiving more research interest. An IDS is a system that passively monitors specific computing resources, such as applications, a machine, or a network, and reports any anomalous activities or any well-known intrusive patterns. Although certain research progress has been made in the field of intrusion detection, it is still a comparatively young area. The current detection algorithms are far from being mature, and are in strong need of thorough testing. Unfortunately, there does not exist an environment which allows researchers to thoroughly test their detection algorithms.

In this thesis, a software platform, named IntruDetector, is designed and implemented. IntruDetector can be used to test intrusion detection algorithms. The testing can be performed in a real environment with a wide range of intrusive cases without any danger of compromising normal system operations. This is achieved by using the technique of hybrid simulation. The data of normal system activities are directly collected from a live environment, and intrusive activities are simulated by
real simulation or virtual simulation. The architecture of IntruDetector is designed based on the Common Intrusion Detection Framework (CIDF), the current ongoing IDS standards. IntruDetector consists of four main components, event generators, an event buffer, an analysis engine, and a visualization module. Event generators are composed by a packet sniffer and simulators. The packet sniffer produces normal events by capturing and processing network packets transmitted over the network. Simulators can be used to generate intrusive events. New simulators can be easily added by users. All events are sent to a central repository, the event buffer. The analysis engine encompasses a detection algorithm. It retrieves events from the event buffer, analyzes them, and reports results to the visualization module. Analysis engines containing other detection algorithms can also be incorporated in the platform. IntruDetector ensures that all analysis engines have a consistent view of events, so their analysis results can be compared on the same basis.

The main contributions of this thesis are: (1) A software platform, IntruDetector, is designed and implemented. It can be used as an environment to test network intrusion detection algorithms. (2) The technique of hybrid simulation is utilized to generate normal and intrusive events. It allows the testing to be performed in a real environment with a wide range of intrusive activities, but without any danger of compromising normal system operations.
Acknowledgments

I would like to express my sincerest gratitude to my supervisor, Dr. Xue Dong Yang, who gave me valuable advice, countless encouragement and generous financial support originating from the Natural Sciences and Engineering Research Council of Canada. These have greatly contributed towards the completion of this thesis.

I am grateful to my internal committee members, Dr. Larry Saxton and Dr. Yiyu Yao, and my external examiner, Dr. Luigi Benedicenti, for their constructive comments and suggestions. The thanks also go to the Department of Computer Science and Graduate Studies of University of Regina for their financial support.

I would like to thank Mr. Qiang Hu for proof reading my thesis, and Chong Li, Tao Wang for their friendship and advice.

I would also like to thank my parents, my brothers and sisters, especially my oldest brother, Fayang Wan, for their continuous support and encouragement through all of my life. Special thanks to my wife, Jing Chen. She gave up her job in China, and joined me in Canada. Her support is crucial to any of my academic achievements.
## Contents

Abstract ................................................. i

Acknowledgments ....................................... iii

Table of Contents ...................................... iv

List of Figures ......................................... viii

Chapter 1  Introduction ................................ 1
  1.1 Computer Intrusion Problems ..................... 2
  1.2 Intrusion Detection Systems ...................... 3
  1.3 Objectives of the Current Research .............. 4

Chapter 2  Review of Previous Work .................. 9
  2.1 The History of Intrusion Detection Systems ...... 9
  2.2 Classification of Intrusion Detection Systems .... 11
    2.2.1 Anomaly Detection versus Misuse Detection ... 11
    2.2.2 Host-based versus Network-based ............. 13
  2.3 Testing of Intrusion Detection Systems .......... 14
Chapter 4  The Design of IntruDetector

4.1 Event Generators .................................................. 37
  4.1.1 Data Structure of Events .................................. 37
  4.1.2 Normal Events Generation ................................. 40
  4.1.3 Intrusive Events Generation ............................... 47

4.2 Event Buffer .................................................. 49
  4.2.1 Data Structures .................................................. 50
  4.2.2 Process Communication and Control ....................... 54
  4.2.3 Application Programming Interfaces ....................... 57

4.3 Analysis Engine .................................................. 61
  4.3.1 Events Assembler .............................................. 61
  4.3.2 Analysis Algorithms ......................................... 63

4.4 Visualization Module ............................................ 65

Chapter 5  The Implementation of IntruDetector and Experimental Results

5.1 Implementation .................................................. 66
  5.1.1 Programming Language Consideration ....................... 66
  5.1.2 User Interface Implementation .............................. 67
  5.1.3 Event Generators Implementation ........................... 67
  5.1.4 Event Buffer Implementation ................................. 68
  5.1.5 Analysis Engine Implementation ............................. 70
  5.1.6 Visualization Module Implementation ....................... 72

5.2 Experiments .................................................. 76
List of Figures

3.1 Real Simulation .................................................. 23
3.2 Virtual Simulation .................................................. 25
3.3 Structure of Hybrid Simulation .................................. 28
3.4 The System Architecture of IntruDetector ..................... 30
4.1 A TCP/IP Packet .................................................. 37
4.2 Ethernet Header ................................................... 38
4.3 IP Header .......................................................... 38
4.4 TCP Header ........................................................ 38
4.5 Comparison of OSI model and TCP/IP protocol suite. ......... 40
4.6 Data Encapsulation of TCP/IP ................................... 41
4.7 How to Capture Packets ......................................... 42
4.8 Packet Sniffer, Libpcap and OS packet capture facility. .... 43
4.9 TCP Three-way Handshake Protocol. .......................... 44
4.10 User Datagram Protocol (UDP) Header. ....................... 45
4.11 First 3 fields of Internet Control Message Protocol (ICMP) .... 46
4.12 A Circular Queue ................................................ 50
4.13 Event Buffer Architecture ...................................... 51
<table>
<thead>
<tr>
<th>Section Number</th>
<th>Section Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.14</td>
<td>Data Structure of An Item</td>
<td>52</td>
</tr>
<tr>
<td>4.15</td>
<td>Data Structure of Index Segment</td>
<td>53</td>
</tr>
<tr>
<td>4.16</td>
<td>Data Structure of Statistical Segment</td>
<td>54</td>
</tr>
<tr>
<td>4.17</td>
<td>Interprocess Communication By Shared Memory</td>
<td>55</td>
</tr>
<tr>
<td>4.18</td>
<td>Algorithm for Event Generators</td>
<td>57</td>
</tr>
<tr>
<td>4.19</td>
<td>Algorithm for Analysis Engines</td>
<td>58</td>
</tr>
<tr>
<td>4.20</td>
<td>Hierarchy of Programming Interfaces</td>
<td>58</td>
</tr>
<tr>
<td>4.21</td>
<td>Network Traffic Hierarchy</td>
<td>62</td>
</tr>
<tr>
<td>5.1</td>
<td>The Interface of IntruDetector</td>
<td>68</td>
</tr>
<tr>
<td>5.2</td>
<td>Packet Sniffer Implementation</td>
<td>69</td>
</tr>
<tr>
<td>5.3</td>
<td>Linked List for Traffic Hierarchy</td>
<td>70</td>
</tr>
<tr>
<td>5.4</td>
<td>Distribution of Services</td>
<td>71</td>
</tr>
<tr>
<td>5.5</td>
<td>Machines Which Are Generating Network Traffic</td>
<td>73</td>
</tr>
<tr>
<td>5.6</td>
<td>Communication Paths Initiated by a Machine</td>
<td>74</td>
</tr>
<tr>
<td>5.7</td>
<td>Service Paths Between a Pair of Machines</td>
<td>75</td>
</tr>
<tr>
<td>5.8</td>
<td>Experimental Network</td>
<td>77</td>
</tr>
<tr>
<td>5.9</td>
<td>The features of Common TCP services in Historical Profile</td>
<td>78</td>
</tr>
<tr>
<td>5.10</td>
<td>Distribution of the Services from Client C/I to Server S/I</td>
<td>79</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In August 2000, a supercomputer at the University of Regina, Canada was successfully broken into by an unknown hacker from a machine in the United States. The hacker created an account with root privilege on the supercomputer without password protection, stole the password file, and installed a program which can be used to launch attacks against other machines. As a consequence, the supercomputer had to be rebuilt from scratch, and all users were forced to change their passwords. It took more than one week for the machine to be back in service again.

This is only one of many attacks that are occurring on the Internet. As the usage of the Internet is proliferating, more and more of our daily lives and business operations depend on the availability of networks and the integrity of data. Unfortunately, the information infrastructure on which we depend is very vulnerable to attacks. Computer intrusions are happening from day to day, and have become a threat to our society. In this chapter, we overview the computer intrusion accidents, describe the systems for detecting intrusions, and present our research objectives.
1.1 Computer Intrusion Problems

An intrusion can be defined as any set of actions that attempt to compromise the integrity, confidentiality or availability of a resource [24]. Computer Intrusions can be divided into six main types [51]: attempted break-ins, masquerade attacks, penetration of security control systems, leakage, denial of service, and malicious use. Many intrusion accidents have occurred during the last two decades: the following are some examples of them.

In 1988 a program known as Internet Worm caused the Internet to be unavailable for about five days [53]. Also in 1988, a document related to the administration of the Enhanced 911 (E911) emergency system was stolen from a BellSouth computer [17].

During the 1980s, Kevin Mitnick, a veteran hacker, broke into numerous computers at Digital Equipment Corporation, Netcom, and other long-distance carriers. He was later convicted of downloading proprietary source codes, and stealing credit card numbers and cellular phone numbers [18].

In the early 1990s, AT&T long-distance service was collapsed as a result of attacks by unknown hackers [43]. On February 1, 1995, unknown attackers severed 7 fiber-optic cables near the Frankfurt/Main airport. About 15,000 telephone lines were interrupted. The cables also carried data for Lufthansa’s booking computers; consequently, new reservations had to be made manually. Lufthansa’s main computers (at Frankfurt airport) were taken out of service for an extended period of time, causing service delays of up to 30 minutes [43]. In the late 1990s, many US federal Web sites, including those of White House, Interior Department, Energy Department, and National Park Service, were broken into [5, 6].
At the beginning of the new millennium, several famous commercial sites (for example, Microsoft, Yahoo, eBay, among others.) were attacked and out of service for several hours [1, 2].

1.2 Intrusion Detection Systems

To protect computing resources from being penetrated, many technologies have been proposed, such as cryptography, authentication, firewall, among others. Recently, Intrusion Detection System (IDS) has been receiving more and more attention.

An IDS is a system which passively monitors specific computing resources, such as applications, a machine, or a network, and reports to System Security Officer (SSO) any anomalous events or any well-known patterns indicating potential intrusions. Compared to a firewall, which tries to prevent intrusions from occurring, an IDS tries to detect intrusions when they are occurring or after they have occurred.

There are two detection techniques which are widely used in the intrusion detection community. They are misuse detection (also called signature analysis) and anomaly detection. Misuse detection tries to identify intrusions by comparing current system activities with the patterns of well-known attacks, and generates an alarm whenever there is a match. Anomaly detection tries to profile the normal behaviors of objects, such as users, groups, files or a system. The behaviors of current system activities are compared to their normal behaviors. If the behaviors of the current system are outside the scope of normal behaviors, the system activities are flagged as anomalous. Anomaly detection is also called statistical analysis, as it uses statistical techniques.
Significant research efforts have been involved in the field of intrusion detection since James P. Anderson introduced the idea of automated intrusion detection in 1980 [9]. Many intrusion detection systems [38, 25, 44] have been developed. Technologies in other fields, such as artificial intelligence [22, 39, 31], information retrieval [10], mobile agents [28], data mining [33], among others, have also been applied to intrusion detection systems. More and more organizations are planning to install IDSs. According to the report [3] by Aberdeen Group, the market for intrusion detection products hit 100 million dollars in 1998, and was doubled in 1999.

1.3 Objectives of the Current Research

Although certain research progress has been made in the field of intrusion detection, it is still a comparatively young area with many open issues.

Firstly, the theoretical foundation is still under development and far from being mature. Most of the current IDSs are based on the Denning model [19]. Although some new technologies, such as information retrieval [10], mobile agents [28], data mining [33], among others, have been introduced to the field, there is no significant improvement of the model since it was first proposed in 1988.

Secondly, this field lacks widely-accepted standards which are adopted by different IDS vendors. Some efforts have been made on the standardization. To promote interoperability among existing intrusion detection systems and provide guidance for future development, Internet Engineering Task Force (IETF) has formed an Intrusion Detection Working Group (IDWG) to develop a set of functional requirements and
protocols for communication between IDSs [46]. The Defense Advanced Research Projects Agency (DARPA) has also formed a group to develop a Common Intrusion Detection Framework (CIDF) [21]. Despite these significant efforts, so far only Event Monitoring Enabling Responses to Anomalous Live Disturbances (EMERALD) [44], developed at Stanford Research Institute (SRI) International, is claimed to conform to any of these standards.

Thirdly, no system environment exists for experimenting and testing intrusion detection algorithms. As intrusion detection is a comparatively young area, many technologies from other fields have been applied and various detection algorithms are under development. Naturally, researchers strongly need an environment for testing and evaluating their systems [55].

The scope of this thesis is restricted to the third issue. The difficulties of testing an intrusion detection system lie in how to generate the data of normal system activities which are sufficiently realistic, and how to generate as many intrusive activities as possible.

Significant efforts have been made in the testing and evaluation of intrusion detection systems [48, 16, 20, 36]. In [48, 16, 20], the environments for testing IDS are proposed. The tests are all conducted in isolated experimental networks. The normal system activities and intrusions are generated in experimental networks by using scripts created with software tools, such as Expect, Tcl, Perl, among others. The above methods suffer some serious drawbacks.

1. The data of normal system activities generated are not realistic or sufficient.

2. The number of simulated intrusions is limited, and the intrusions which are
destructive to the experimental environment can not be simulated.

A comprehensive study for comparing the performance of different IDSs was conducted by MIT’s Lincoln Laboratory in 1998 [36]. The laboratory distributed the testing data to the researchers of IDSs to conduct off-line testing and evaluation. The data are generated by recording system activities from a military base and replicating them in an experimental environment. The data represent both normal system activities and attacks occurred in the real environment or generated by simulation. The corpus of data is the most comprehensive set known to date. The data are not only used by researchers in that specific evaluation, but also in their daily experiments of testing. The weakness of the data is that intrusive data are far from sufficient and there is no way for researchers to add new intrusive data. Some researchers have experienced difficulties with the data in their testings [23].

This thesis focuses on creating a software platform for testing network intrusion detection algorithms. The software platform is named IntruDetector. IntruDetector is developed by applying the techniques of hybrid simulation. A hybrid simulation system can sample data from a real environment, and at the same time, allows virtual data to be generated. The architecture of IntruDetector is designed based on the CIDF standards. IntruDetector consists of four main components: event generators, an event buffer, an analysis engine, and a visualization module. Event generators are composed by a packet sniffer and real and virtual simulators. The packet sniffer collects the data of normal system activities directly from a live environment by capturing and processing network packets transmitted across broadcast Local Area Network (LAN). Intrusions are simulated by real or virtual simulators. Simulators are
user configurable, and new simulators can be easily added by users. Intrusive data, mixed with normal data, are sent to a central repository, the event buffer, where the mixed data will be retrieved by the analysis engine. Other analysis engines can also be incorporated. All analysis engines are ensured to have a consistent view of events, so their analysis results are comparable on the same basis. The visualization module presents the analysis results to a graphic user interface, and can visualize network traffic intuitively. The main features of IntruDetector are:

- The generated data of both normal system activities and intrusions are sufficient and realistic. Normal data are collected directly from a real environment, and intrusive data are generated by simulation.

- By using the technique of hybrid simulation, a variety of intrusion types can be simulated, including certain types of destructive intrusions which may not be suitable for real simulation. It also frees us from any danger of compromising normal system operations when simulating intrusions in a real environment.

- It is highly extensible and flexible. New simulators and analysis engines can be easily added. Researchers can develop simulators to simulate intrusions of their interest, and different detection algorithms can be tested in the environment simultaneously.

It should be mentioned that IntruDetector is designed based on Ethernet and UNIX environment for preliminary study, and it does not support other types of networks (for example, FDDI and ATM) at the moment. It will also require further development to migrate IntruDetector from UNIX to Windows.
The rest of the thesis is organized in the following manner. Chapter 2 will review the previous work in the field of intrusion detection. A brief history of intrusion detection systems will be presented, and main techniques used by IDSs will be discussed.

In Chapter 3, we will introduce the important concepts which will be helpful to the understanding of our system. The system architecture will also be discussed in this chapter.

The design of the main components of IntruDetector, including event generators, the event buffer, the analysis engine and the visualization module, will be presented in Chapter 4.

In Chapter 5, we will describe the system implementation and some experiments we have conducted with IntruDetector.

The last chapter summarizes the current research, and highlights the main contributions. Possible topics for future research are also discussed.
Chapter 2

Review of Previous Work

2.1 The History of Intrusion Detection Systems

The earliest work of intrusion detection systems was a study by James P. Anderson [9]. He categorized possible intrusions into four groups: external penetrator, misfeasor, clandestine user, and masquerader. In his report, Anderson suggested that system audit trail records could be used to detect intrusions. Some intrusions could be detected by looking for specific patterns of the activities, and others such as masquerader could be identified by looking for their deviations from the user’s normal behaviors. This is the first piece of work in the literature that presents the idea of automatic intrusion detection in computer systems.

In 1987, Dorothy Denning [19] presented a model for a real-time intrusion detection system that aims to detect a wide range of security violations. This was the first time that an intrusion detection system was proposed as a solution to computer security problems. Since that time, many research efforts have been devoted to intrusion detection systems.
In 1938, Teresa Lunt et al. refined the model proposed by Denning and created the Intrusion Detection Expert System (IDES) [38] at Stanford Research Institute (SRI). In 1988, Haystack [51] was developed to detect intrusions in a mainframe environment at the US Air Force Bases. In the same year, MIDAS [50] was developed by the National Computer Security Center, in cooperation with the Computer Science Laboratory, SRI International. In 1989, Wisdom & Sense [55] was introduced by Los Alamos National Laboratory to detect anomalous computer session activities.

In 1990, Network Security Monitor (NSM) [25, 41] was developed at the University of California at Davis. NSM is the first system which detects intrusions by analyzing network traffic, instead of system audit trail records. In 1991, Network Anomaly Detection and Intrusion Reporter (NADIR) [26, 27] was developed at Los Alamos National Laboratory. In 1992, DIDS (Distributed Intrusion Detections System) [52] was introduced. The differentiator of DIDS is that it integrates Haystack [51] and NSM [25, 41] together, and addresses the question of how to handle distributed, heterogeneous systems [11]. In 1994, IDIOT [32] was developed at the Department of Computer Science, University of Purdue. The authors suggested that a layered approach should be used when pattern-matching-based techniques are applied to the problem of intrusion detection. In the same year, Mark Crosbie and Eugene Spafford proposed to apply autonomous agents techniques to IDS, in order to improve the scalability, maintainability, efficiency, and fault tolerance of IDS [15].

In 1996, GrIDS (Graph based Intrusion Detection System) was introduced to detect large-scale automated or coordinated attacks [13]. In 1997, EMERALD [44] was introduced by SRI International. EMERALD was built on, and considerably
extended many years of research in, intrusion detection systems at SRI International. In 1997, JiNao [30] was developed at North Carolina State University to protect distributed network infrastructure, instead of individual hosts on a network. In 1998, the techniques of information retrieval were suggested for IDS [10].

2.2 Classification of Intrusion Detection Systems

An intrusion detection system (IDS) analyzes system and user activities in search of undesirable patterns from a security perspective. Data sources for intrusion detection may include audit trials produced by an operating system, network traffic flowing between systems, or application logs. Analysis can be conducted off-line or in real-time. There are two main approaches to classifying intrusion detection systems. One is based on the detection techniques, anomaly detection or misuse detection; the other is based on the targets monitored by an IDS, host-based or network-based.

2.2.1 Anomaly Detection versus Misuse Detection

2.2.1.1 Anomaly Detection

Anomaly detection approach is based on the hypothesis that any intrusive activity will be noticeably different from normal activities. Therefore, it can be detected by looking for abnormal patterns of system usage [19].

In this method, historical profiles are created for the subjects of interest, such as files, directories, users, network protocols, among others. Current system activity is compared with historical profiles to determine if it is outside of the scope of normal
behavior. Alarms will be signaled when observed values fall outside of the normal range. For example, an intrusion could be reported if a telnet session was observed which was initiated from the outside by a user who always logs in from the inside.

The main issues in anomaly detection are:

- How to select system features to monitor.
- How to build profiles representing the normal behavior of selected features.
- How to select threshold values to keep the number of false alarms at an acceptable level.

The advantages of anomaly detection are [12]:

- The system may detect heretofore unknown attacks.
- The system may detect complex attacks, such as those that occur over extended period of time.

The disadvantages of this approach are:

- It is possible for an attacker to trick the system to accept intrusive activity as normal.
- The changes in user behaviors may not be handled properly.

2.2.1.2 Misuse Detection

Misuse detection is very similar to the idea of a virus detection system. It is based on the assumption that each attack has an associated pattern or signature which can be used to identify the attack.
The advantage of misuse detection systems is that they may detect many or all known attack patterns. The drawback is that they are of little use for as yet unknown attack methods. The main issues in misuse detection systems are:

- How to write a signature that encompasses all possible variations of a specific attack, yet excludes non-intrusive activities.
- How to build a signature database that includes the signatures of as many known attacks as possible.

2.2.2 Host-based versus Network-based

2.2.2.1 Host-based IDSs

Host-based IDSs depend on audit trail records generated by host operating systems as the main information sources. The advantages of this approach are:

- Specific system activities can be monitored, such as file accesses, login and logout activities, and what each user does during one login session.
- It works in encrypted or switched environments which are proliferating.

The disadvantages are:

- Host-based IDSs are system dependent. Since most operating systems have their own mechanisms for producing audit trail records, host-based IDS developed for one Operating System (OS) can not be deployed in other OSs.
• Host-based IDSs themselves are vulnerable to attacks. If the mechanism for producing audit trail records is disabled, the data source of Host-based IDSs will be completely unavailable.

2.2.2.2 Network-based IDSs

Network-based IDSs detect attacks by analyzing network traffic transmitted across the network. The information is usually gathered by a packet sniffer which listens to a network interface set into the promiscuous mode. The strength of this method is that it is system independent, and can detect a wide range of sophisticated intrusions. The weakness is that it is useless in switched or encrypted environments.

2.3 Testing of Intrusion Detection Systems

Like any other software, Intrusion Detection Systems need extensive testing before they can become mature. Some efforts in testing IDSs have been made by the researchers at UC Davis, MIT Lincoln Labs, and IBM Zurich Research Laboratory.

2.3.1 Testing at UC Davis

The researchers at the University of California at Davis have been the first to address the issues of testing intrusion detection systems. They have developed a methodology and software platform for testing IDSs. UNIX packages, for example, Expect [34], was employed to develop the scripts for simulating intruders as well as normal users. Intruders can be individual or cooperative. Three testing objectives
[47] are identified as follows:

- Broad Detection Range: the ability to distinguish intrusions from normal activities.

- Economy in Resource Usage: the efficiency of using system resources.

- Resilience to Stress: the ability to function correctly under high load

Three testing procedures corresponding to the above objectives were developed, and Network Security Monitor (NSM) [25] was evaluated in the environment as an example.

The drawback of this methodology is that attacks simulated by the scripts written in Expect will probably interrupt normal system operations. For example, if a denial of service attack was simulated, the target system under the attack may be out of service, and normal users will lose their accesses to the machine.

2.3.2 Testing at MIT Lincoln Labs

MIT Lincoln Labs conducted two IDS testings in 1998 and 1999 respectively [36, 7]. The testings were not performed in a real environment. Instead, a simulated network was constructed as the testing bed. The testing data were sampled from the real environment, the US Air Force Bases. In order to protect privacy and confidentiality, all sensitive information such as user names, IP addresses, among others, were pseudonymized. The pseudonymized data were replayed in the simulation network by manually duplicating transactions that occurred in the US Air Force Bases.
Several well-known attacks were simulated, and inserted into normal system activities. Several major IDSs were tested, and the testing results were also published.

This testing is very realistic, and can be used as a basis to compare different IDSs. The disadvantages are that the workload of duplicating normal system activities is tremendously high, and the cost of the testing is prohibitively expensive. This would be regarded as being impractical by most other IDS researchers.

2.3.3 Testing at IBM Zurich Research Laboratory

IBM Zurich Research Laboratory has developed an experimentation workbench for testing intrusion detection systems [16].

First, an experimentation network was constructed, which was disconnected from the rest of the world. Both normal user behaviors and intrusions are obtained by simulation. Normal user behaviors are simulated by scripts written in Expect. Intrusions are simulated by the exploiting scripts from a vulnerability database which is maintained by the lab internally. After examining the analysis results of various IDSs, the operator can therefore evaluate their performance.

The researchers have been aware that the workbench must cope with a heterogeneous environment, and both normal user activities and attacks should be generated in a "portable" way. Unfortunately, they did not follow any IDS standards. Therefore, it is difficult to test in the workbench the IDSs from different vendors.
2.4 Standards of Intrusion Detection Systems

In order to promote interoperability amongst Intrusion Detection Systems, some efforts have been made to develop the standards for intrusion detection systems.

2.4.1 Effort by DARPA

DARPA is pursuing an effort to develop a Common Intrusion Detection Framework (CIDF). The CIDF effort focuses on establishing a high level architecture of IDS functional components, and a common language between these components.

CIDF envisages Intrusion Detection Systems as a set of discrete components which communicate via message passing. These components include:

- Event Generators (E-boxes): they produce events.
- Event Analyzers (A-boxes): they consume events, and generate events representing the analysis results.
- Event Databases (D-boxes): they store events for later retrieval.
- Response Units (R-boxes): they consume events, and carry out actions.

All events produced and consumed by the IDS components are in the same form of Generalized Intrusion Detection Objects (GIDOs). GIDOs are defined by a standard common format, Common Intrusion Specification Language (CISL).

The significance of CIDF is that it makes it possible for different IDSs to be integrated together and to share their information to detect complex and large scale attacks.
2.4.2 Effort by IETF

Encouraged by the ideas involved in CIDF, Internet Engineering Task Force (IETF) has formed the Intrusion Detection Working Group (IDWG). IDWG is working on the definition of the standards of data formats and exchange procedures for intrusion detection systems. Several documents of Intrusion Detection Exchange Format have been published, and are being considered by Instructional & Electronics Support Group (IESG) as RFCs. Because the effort by IETF was inspired by the ideas of CIDF, and the members of IDWG overlap with those of CIDF working group, this standard shares some commonality with CIDF.
Chapter 3

Conceptual Framework and System Architecture

The objective of this research is to develop a software platform which can be used to test network intrusion detection algorithms. In order to achieve the above objective, we design the system by following the current ongoing standards of intrusion detection systems, and utilize the technique of hybrid simulation to generate normal and intrusive data. In Section 3.1, we describe the system requirements of the platform for testing intrusion detection algorithms. In Section 3.2, the technique of hybrid simulation is presented. The system architecture is discussed in Section 3.3. In Section 3.4, we demonstrate how the system operates.
3.1 System Requirements

According to Common Intrusion Detection Framework (CIDF) [46], the current ongoing IDS standard, an intrusion detection system (IDS) is divided into four components: event generators (E-boxes), event analyzers (A-boxes), event databases (D-boxes), and response units (R-boxes). Each component has its distinct functionality, and communicates with others by message passing. Among these components, event analyzers are the core of an IDS. In this thesis, event analyzers are also referred to as analysis engines or detection algorithms, as an analyzer might use detection algorithms to perform the analysis. Based on the architecture of an IDS, we envisage that the software platform for testing detection algorithms should consist of the following components:

- Event generators. Event generators are responsible for producing events and providing them to the detection algorithms. If an event generator obtains events from a real computational environment, its inputs could be application logs, operating system audit trails, network traffic, among others. If an event generator produces events based on a statistical model or a predefined process, the inputs might be configurable parameters. The outputs of an event generator are events representing real or simulated system activities.

- Analysis engines. The role of an analysis engine is to analyze events by using various algorithms and return results presumably representing some kind of synthesis or summary of the input events. The inputs to an analysis engine are events produced by event generators, and the outputs are its analysis results.
The performance of a detection algorithm can be measured by evaluating its outputs. If a detection algorithm can correctly identify intrusions with comparatively low false alarms, its performance is regarded as being efficient. Many detection algorithms, which are under development by utilizing the technologies from other fields, are in strong need of thorough testing.

- Presentation Module. The presentation module is to show analysis results of detection algorithms intuitively. It may also consist of a graphical user interface which simplifies the application of the platform.

- Mechanisms for communication. The outputs of the event generators are the inputs to the analysis engines and the outputs of the analysis engines will be presented by the presentation module; there should exist mechanisms for these components to communicate with each other.

### 3.2 Hybrid Simulation

Computer simulation plays a fundamental role in many research fields. It is utilized for a variety of different reasons, mainly because either the cost or risk is prohibitively high, or it is impossible to do an experiment in a real environment. Although simulation has been widely applied in many areas of computer science, such as system design, routing algorithm development, performance evaluation, among others, very little work has been done in the field of intrusion detection.

This thesis proposes a framework for testing intrusion detection algorithms by using the technique of hybrid simulation. To perform the testing, the data of both
normal and intrusive system activities should be generated. There are several different ways to generate them. These methods are summarized as follows.

3.2.1 Normal System Activity Generation

The data of normal system activities are needed for two reasons. First, some anomaly detection systems need them as training data to create profiles of normal system behaviors. Second, they are needed as the background traffic when intrusion detection systems are tested. There are three methods for generating the data of normal system activities.

3.1.2.1 Real Data

The data of normal system activities can be collected directly from a real environment. We call the data from a live environment the real data. Real data will be processed by programs, called data filters, which convert the raw data (for example, audit trails, network packets, among others.) into a canonical form that can be understood by analysis engines. Data filters also reduce the volume of data to a certain degree. In order to obtain the real data, we can either conduct the testing in a real environment, or transfer the real data to an experimental environment. Unfortunately, neither of these two methods is easy.

Performing the testing in a real environment could compromise normal system operations, as intrusive activities are also needed to be generated in the real environment. If the testing is conducted in an experimental environment, how can the data from the real world be duplicated there? The method deployed by MIT Lincoln Labs
is to employ several persons to do, in the experimental environment, jobs similar to those that occur in the real environment. For example, in order to simulate FTP traffic, several users downloaded a variety of source codes and documents files from anonymous FTP sites inside and outside. The data generated in this way are realistic, but the work load is heavy and the expenses are high. Therefore, it is impractical for most other researchers.

3.1.2.2 Real Simulation

Since it is dangerous to perform the testing in a real environment and difficult to duplicate the data from a real world in an experimental environment, some researchers [48, 16] try to generate normal system activities by running simulating programs in experimental environments. We refer to these programs as the real simulatons as they generate the real data, such as audit trails, network packets, among others. This is illustrated by Figure 3.1.

![Figure 3.1: Real Simulation](image)

This method was adopted by the researchers at UC Davis [48] and IBM Zurich
Research Laboratory [16]. They used software tools such as Expect, Tcl, Perl, among others, to create programs for generating normal traffic. For example, in [48], a program was written using Expect to simulate a user's normal activities during a login session, such as changing directories, listing directories, editing files, among others.

In [16], a program was developed to simulate the behavior of the FTP daemon.

The advantage of real simulation lies in its simplicity. The simulation programs are easy to develop, and can be run whenever necessary. The drawback is that simulated data may not be sufficient or realistic.

3.1.2.3 Virtual Simulation

The third way to generate normal system activities is completely different from the above two methods. We call this method virtual simulation since no real system data (for example, audit trails, network traffic, among others.) are generated.

Virtual simulation simulates system activities by generating events to be fed to analysis engines directly, instead of generating real traffic or audit trails which will be processed by data filters first. Figure 3.2 depicts the scenario.

To virtually simulate normal system activities, we need to collect and analyze data from a real environment, study their statistical models, and develop virtual simulators which can generate events conforming to the distributions of the statistical models. Unfortunately, we do not know exactly which statistical model the distribution of system activity data is close to. Most researchers assume that data is Gaussian distributed, but [25] shows that the distribution is generally multi-modal.
3.2.2 Intrusive System Activity Generation

Intrusive activity should be generated and mixed with normal system activity when testing IDSs. Intrusive activity can be generated by real simulation or virtual simulation. It is also possible to collect real data of intrusive activity, but the number of types of attacks collected may be limited.

3.1.3.1 Real Simulation

Software tools or programming language can be used to develop exploitive programs which can explore a specific system vulnerability and launch attack. Intrusive system activities can be generated by running such exploitive programs. We use SYN-Flooding denial of service attack as an example to illustrate.

SYN-Flooding is a denial of service attack. It explores a weakness of the TCP/IP protocol. In TCP/IP, a three-way handshake protocol is used to establish a connection. An initiator sends out a SYN segment to a receiver to request for establishing a connection. After receiving the SYN packet, the receiver will send back a ACK
segment to acknowledge it, and at the same time, some memory space is allocated to store the status information of the connection being established. Normally, a initiator will send back to the receiver another ACK segment to complete the establishment of the connection.

In SYN-Flooding attack, the initiator sends out a huge number of requests for connection establishment in a very short period of time, but does not send back ACK segments. In this case, the receiver will keep the status information of all uncompleted connections until time-out. If the number of uncompleted connections exceeds the system limit, the receiver may run out of memory, and legitimate requests will be dropped. In the worst scenario, the receiver could be forced to shut down.

We can write a program to launch this type of attack. The program will open a raw socket, compose SYN packets, and send them out to a target. To spoof the target, the sources of SYN packets can be set to any arbitrary addresses. In this way, the ACK segments sent back by the target will be lost and the target will never be able to receive any acknowledgments. If the number of uncompleted connections kept by the target exceeds its limit, the target will not be able to accept any legitimate requests. In the worst scenario, the target will have to be brought down.

The strength of real simulation is that simulated results are sufficiently realistic. A drawback is that destructive attacks cannot be simulated in a real environment, since this could bring serious damages. For example, if we run the program of SYN-Flooding attack in a live environment, legitimate users may lose their access to the machine being attacked.
3.1.3.2 Virtual Simulation

The second way to generate intrusive activity is by virtual simulation. We will still use the SYN-Flooding attack as an example. Recall that in real simulation, the exploitive program will send out huge number of SYN packets to a target in a very short period of time. The data filter will generate an event of TCP connection establishment request for each detected SYN packet. The events will be stored in event databases where analysis engines will retrieve and analyze them.

The virtual simulation program will not generate any real traffic; instead, it will generate TCP connection establishment events directly, and those will be fed to analysis engines. Although no real data are produced, simulated events are still realistic as long as they are comparable to the events generated by the data filter when the real simulator is running.

Virtual simulation is very helpful when destructive intrusions need to be simulated. By using virtual simulation, testing of intrusion detection algorithms can be conducted in a real environment, but without the danger of compromising normal system operations.

3.2.3 Hybrid Simulation

Based on the methods described above, we propose a combined approach to generating the data of normal and intrusive system activities. It is called hybrid simulation. Data of normal system activities are collected from live environments, and intrusive system activities can be simulated by real or virtual simulation. Virtual simulation can be deployed when destructive intrusions (for instance, denial of services) are
simulated. To simulate non-destructive intrusions (for instance, protocol scanning), both types of simulation can be used. It is possible that the data of normal system activities contain intrusive data. This is out of our concern as we want to test if simulated intrusions can be successfully identified. The structure of hybrid simulation is illustrated by Figure 3.3.

The advantages of hybrid simulation are:

- Background data are realistic and sufficient since testing is conducted directly in a real environment.

- A broad range of intrusions can be simulated by using the technique of hybrid simulation.

- Normal system operation is protected since destructive intrusions can be simulated by virtual simulation.
3.3 The System Architecture

We designed the software platform, named IntruDetector, for testing network intrusion detection algorithms based on the requirements described in Section 3.1. IntruDetector consists of several components. They are event generators, an event buffer, an analysis engine, and a visualization engine. IntruDetector also provides Application Programming Interfaces (APIs) for the communication between simulators and analysis engines. The system architecture of IntruDetector is depicted by Figure 3.4.

3.3.1 Event Generators

Event Generators include Data Filters and Virtual Simulators. Data Filter processes real data and produces corresponding events. Real data could be network packets, system audit trails or application logs. This thesis focuses on building a platform for studying network-based IDSs, so the data filter is actually a packet sniffer which captures network packets and generates network events based on the content of packets. Each packet consists of headers and data. To protect privacy and confidentiality, only the headers of a packet are analyzed. Packets are captured by calling packet capture functions provided by the library, libpcap [40].

It should be pointed out that the platform does not impose any limitations on the model for testing host-based intrusion detection systems. Since the model itself is extensible, both types of intrusion detection systems can be encompassed. Virtual Simulators are developed based on the need of the experiment.
3.3.2 Event Buffer

The event buffer is a central repository where all events are stored. Events can be produced by the packet sniffer or by simulators. Analysis engines will retrieve events from the event buffer and perform analysis. An event will not be removed from the event buffer until being consumed by all analysis engines. This is to ensure that all analysis engines have a consistent view of events and their analysis results can be compared on the same basis.
The event buffer is actually a central buffer used by multiple producers and multiple consumers. Event generators, including the packet sniffer and simulators, are producers, and analysis engines are consumers. To improve system performance, each producer or consumer is represented by a process in the operating system. To achieve maximum efficiency of interprocess communication, the event buffer is implemented using shared memory. Processes are synchronized using semaphores.

3.3.3 Analysis Engine

IntruDetector can be used to study different detection algorithms. In this thesis, a network intrusion detection algorithm is implemented to demonstrate the applicability of the platform. The detection algorithm is referred to as an analysis engine. It is based on the idea of Network Security Monitor (NSM) [25] developed at University of California at Davis. The analysis engine is composed by an event assembler and two analysis algorithms, statistical and signature analysis.

The event assembler is responsible for restoring network events into TCP sessions and organizing sessions into a hierarchical structure. The analysis algorithms are called periodically to check the security status of each TCP session and report nonsecure sessions to the visualization module. Statistical analysis looks for unusual traffic patterns. This requires the knowledge of historical traffic patterns. We collected three weeks worth of network traffic from a subnet at the CS department, and built historical profiles for usual traffic patterns from the data. The lower the probability of a communication path in historical profiles, the more anomalous that communication path is. For example, if a TELNET session from machine A to machine B is very
rare, the occurrence of the connection may indicate an intrusion. Signature analysis
looks for a specific traffic pattern which indicates a known intrusion or violation of
security rules. For example, all accesses to machine A are restricted and can only
come from within the organization. Any connection from outside is a violation of
security rules and will fire an alarm.

3.3.4 Visualization Module

The visualization module will present the results from analysis engine in an intu-
itive way. It will print a message in the message panel when a communication path
is found to be anomalous or in violation of security rules. It can also visualize the
traffic hierarchy constructed by the analysis engine.

3.3.5 System Features

IntruDetector has the following features.

- Realism. Testing of intrusion detection algorithms can be conducted in a real
  environment, and is kept from the danger of compromising normal system op-
erations.

- Applicability. By using the technique of hybrid simulation, a wide range of
  intrusive activities can be simulated.

- High performance. Event generators, including data filters and simulators, and
  analysis engines run concurrently, taking advantage of the power of a multipro-
cessor system. The performance of interprocess communication is significantly
improved by using shared memory since kernel involvement is not necessary.

- Extensibility. Simulators and analysis algorithms can be easily added by users. The templates of simulator and analysis algorithm are also provided. The system can also be extended to support other types of protocols (for example, IPX).

- Flexibility. Simulators can be run at any time without interrupting the system. They can run sequentially or concurrently since each simulator is an independent process.

- Standardization. The design of the platform conforms to a current ongoing standards: CIDF. Event generators are equivalent in function to CIDF E-boxes. The event buffer is similar to CIDF D-boxes. The analysis engines can be regarded as the CIDF A-boxes.

3.4 Operation of the System

We use an example to illustrate how IntruDetector works. Suppose we want to simulate a TELNET communication path from machine A to B, and test how the analysis engine analyzes the connection. Assume that B is being monitored by our system.
3.4.1 Event Generation

This connection can be generated by real simulation or virtual simulation. In real simulation, we can issue a TELNET command on machine A and connect to B. This requires that we have access to machine A. If that is not the case, real simulation will not be applicable. The packet sniffer will be able to detect the connection and generate three types of events. These events will be sent to the event buffer by the packet sniffer.

- TCP connection establishment event
- TCP data transmission events
- TCP connection close event

In virtual simulation, we can develop a virtual simulator to generate these three types of events and send them directly to the event buffer. There is not any difference between the events generated by the packet sniffer and virtual simulator.

Virtual simulation is more flexible than real simulation. First, it does not require that we have access to machine A. Second, it can simulate a connection which occurred at any time. In real simulation, we must either issue the TELNET command at the time of our interest or tell the operating system to run a script at that time.

3.4.2 Event Analysis

The analysis engine will retrieve the events from the event buffer and analyze them. First the events assembler restores the TCP connection based on the events. The information about the connection includes:
• The time that the connection occurred.

• The number of packets transmitted over the connection.

• The number of bytes of data transmitted over the connection.

The statistical analysis algorithm checks the connection with historical profiles. If the TELNET connection from A to B has never occurred before or occurred at a very low probability, it is flagged as anomalous. Otherwise, it is deemed normal. The signature analysis algorithm applies security rules to the connection. If A is not supposed to communicate with B by TELNET, an alarm is triggered.

3.4.3 Event Visualization

The visualization engine visualizes the connection. Histogram and color are used to represent the number of packets and bytes of data transmitted over the connection. An alert message will also be displayed when the connection is flagged as anomalous or intrusive.
Chapter 4

The Design of IntruDetector

As is mentioned in Chapter 3, IntruDetector is composed of four main components: the event generators, the event buffer, the analysis engine and the visualization module. The event generators are responsible for generating events, either from a stream of packets or by virtual simulation. The event buffer manages events in a central repository and ensures that all analysis engines have a consistent view of events. The analysis engine, designed to test the platform and show the applicability of the platform, analyzes events and detects any anomalous or intrusive activities. The visualization module presents analysis results with a graphic user interface and visualizes network traffic intuitively. The design of these four components are described in this chapter.
4.1 Event Generators

4.1.1 Data Structure of Events

In network-based IDSs, events are generated from a stream of packets transmitted over the network. As a result, the data structure of the events is determined by the data structure of packets.

4.1.1.1 The Data Structure of Packets

A packet, as illustrated in Figure 4.1, is composed of two parts, the headers and the data. To protect the users' confidentiality and privacy, only the packet headers are utilized in detecting intrusions. Figure 4.2, 4.3, and 4.4 show the Ethernet, IP, and TCP headers of a packet respectively [14].

![Figure 4.1: A TCP/IP Packet](image)

4.1.1.2 The Data Structure of Events

To encompass the necessary information for detecting intrusions, an event should consist of the following fields:

- Event type: This is the type of the event, such as TCP, UDP, and ICMP.
- Event type subcode: This contains further information on the event type.
- TCP connection identifier: It uniquely identifies a TCP connection.
**Figure 4.2: Ethernet Header**

<table>
<thead>
<tr>
<th>0</th>
<th>48</th>
<th>96</th>
<th>112</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Destination Ethernet Address</strong></td>
<td><strong>Source Ethernet Address</strong></td>
<td><strong>Protocol Type</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4.3: IP Header**

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>19</th>
<th>24</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vers</strong></td>
<td><strong>Header Len</strong></td>
<td><strong>Service Type</strong></td>
<td><strong>Total Length</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Identification</strong></td>
<td><strong>Flag</strong></td>
<td><strong>Fragment Offset</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Time To Live</strong></td>
<td><strong>Protocol</strong></td>
<td><strong>Header Checksum</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Source IP Address</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Destination IP Address</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IP Option</strong></td>
<td><strong>Padding</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4.4: TCP Header**

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>10</th>
<th>15</th>
<th>24</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Source Port</strong></td>
<td><strong>Destination Port</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sequence Number</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Acknowledgement Number</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Header Length</strong></td>
<td><strong>Reserved</strong></td>
<td><strong>Code Bits</strong></td>
<td><strong>Window</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Checksum</strong></td>
<td><strong>Urgent Pointer</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Option</strong></td>
<td><strong>Padding</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
- Initiator: This is the Internet address of the entity initiating the connection.

- Receiver: This is the Internet address of the entity to which the connection was made.

- Initiator port number: This is an integer used to identify the particular application running on the initiator.

- Receiver port number: This is an integer used to identify the particular service used for this connection.

- Start time: This is the time when the event occurred.

- Number of packets: This is the number of packets represented by the event.

- Number of bytes of data: This is the number of bytes of data represented by the event.

Events can be classified based on the types of the protocols denoted in the IP headers. The protocols of interest to us are TCP, UDP, and ICMP, as most of the network traffic in an Ethernet environment uses one of these protocols. We divide events into three categories: TCP events, UDP events, and ICMP events. TCP is a connection oriented protocol, which uses management messages to establish, terminate, and control transport layer connections. Examples of management messages are TCP SYN, TCP RESET, TCP ACK, among others. To differentiate management messages from data messages, TCP events are further divided into management events and data events.
### 4.1.2 Normal Events Generation

Normal events are generated by a packet sniffer which captures and parses network packets. The packet sniffer can capture all packets transmitted over a broadcast Ethernet segment by setting the Ethernet interface to promiscuous mode.

In order to understand how the packet sniffer works, we must first review the operation of TCP/IP over Ethernet. We then describe the fundamental principles of libpcap [40], over which the packet sniffer is built, and finally, we show how packets are processed to generate events.

#### 4.1.2.1 TCP/IP over Ethernet

Figure 4.5 compares a TCP/IP protocol suite with the Open System Interconnection model (OSI) proposed by International Organization for Standardization (ISO). Figure 4.6 demonstrates the process of data encapsulation of TCP/IP.

![Comparison of OSI model and TCP/IP protocol suite.](image)

Figure 4.5: Comparison of OSI model and TCP/IP protocol suite.
Figure 4.6: Data Encapsulation of TCP/IP

As can be seen from Figure 4.6, messages passed between two hosts must go down through several layers in one side and go up across the same layers in the other side. The layering principle allows the protocol designer to focus on one layer at a time, without worrying about how the lower layers perform. A message will be decomposed into packets when it is passed from the application layer to transport layer. A header will be added to a packet when it is passed to a lower layer, and removed when the packet goes up to a higher layer. To obtain the information contained in packet headers, we need to capture packets before the headers are removed. This should be done at the lowest layer space (Ethernet layer). Additionally, capturing all packets transmitted over the network should also be done at the Ethernet layer.

4.1.2.2 The Principles of Libpcap

The Ethernet was built upon the principle that all machines on a local network share the same wire. Messages transmitted to any machine are broadcasted on the wire. This makes it possible for any machine to receive all the traffic on the local cable.
Normally, the Ethernet device only receives traffic destined for it, and ignores all the other traffic. By setting the Ethernet device into promiscuous mode, we can force it to receive all network traffic transmitted over the wire. This provides a mechanism for capturing all network traffic.

Normally, packets received by the Ethernet device will be passed up to the network layer for further processing. In order to monitor network activities, most operating systems provide a facility to get a copy of all received and transmitted packets before they are passed up or transmitted out. Three common methods for capturing packets under UNIX are the BSD Packet Filter (BPF), the SVR4 Data Link Provider Interface (DLPI), and the Linux SOCK_PACKET interface. Figure 4.7 [54] illustrates the working scenario of BPF and DLPI.

![Diagram of packet capture](image)

Figure 4.7: How to Capture Packets

Libpcap is a packet capture library. It provides a system independent access to the underlying packet capture facility provided by an operating system. By using libpcap, the packet sniffer will gain the following advantages.

- Independence from the network link layer technology (Ethernet, FDDI, SLIP,
among others).

- Independence from the operating systems (Linux, IRIX, Solaris, among others).

Figure 4.8 shows the relationship between the packet sniffer, libpcap, the operating system's packet capture facility, and the underlying network.

![Diagram](image)

Figure 4.8: Packet Sniffer, Libpcap and OS packet capture facility.

4.1.2.3 Packet Stream Process

The packet sniffer is responsible for parsing the packet stream and generating events. This requires the packet sniffer to be protocol knowledgeable. The packet sniffer will only process three types of packets: TCP, UDP, and ICMP. Packets of other protocols will trigger exceptional events since they are not expected.

**TCP Packets Processing**

TCP is a reliable and connection oriented protocol. It guarantees that a message
will be delivered successfully to the destination in the original order. It uses a three-way handshake protocol for connection establishment. This is denoted in Figure 4.9. The initiator first sends a TCP SYN segment to the receiver to request to establish a connection. The receiver then sends a TCP SYN segment back to the initiator. Finally the initiator sends back a TCP ACK segment to the receiver for acknowledgment. After that, the connection is successfully established. The connection will be terminated when the data transmission completes.

![Figure 4.9: TCP Three-way Handshake Protocol.](image)

To correctly represent the different steps of TCP communication, three types of events will be generated.

- **TCP connection establishment event.** This event indicates that a connection is being established. A globally unique identifier is generated for the connection.

- **TCP data transmission events.** These events represent the data being transferred over the connection.

- **TCP connection closed event.** This event indicates that a connection is closed.

The contents of the events are obtained from TCP and IP headers. Events will be kept in the cache for a period of time which can be defined by users, and aggregated
based on the connection identifiers. This can significantly reduce the volume of data.
Events will be sent to the event buffer if one of the following situations is satisfied.

- A TCP connection establishment event is generated. Since a new connection is potentially an intrusion, its security status should be checked by the analysis engine as soon as possible.

- The timer expires. Events will be sent to the event buffer periodically. The time interval can be configured by users.

- The cache is full. In this case, events in the cache will be sent to the event buffer, and the cache is cleared.

- A TCP connection closed event is generated. When a connection is closed, the event associated with the connection will be sent to the event buffer and the event is cleared from the cache.

**UDP Packets Processing**

UDP is an unreliable and connectionless protocol. It does not provide a mechanism to guarantee the success of message deliveries or the orders of messages arrival. A UDP header is shown in Figure 4.10.

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Length</td>
<td>Checksum</td>
</tr>
</tbody>
</table>

Figure 4.10: User Datagram Protocol (UDP) Header.

The source port number, destination port number, and message length contained in UDP headers, illustrated in Figure 4.10, are utilized for generating UDP events.
The values of the initiator and the receiver in UDP events are from IP headers. UDP events will be kept in the cache for a period of time which can be defined by users, and aggregated based on the 4-tuple (source IP, destination IP, source port, destination port). Events will be sent to the event buffer after a period of time or when the cache is full.

**ICMP Packets Processing**

Internet Control Message Protocol (ICMP) provides a mechanism for reporting an error of a datagram to its original source, and allows a host to request information from another one. It is a required part of an IP and must be included in every IP implementation. Like TCP and UDP, ICMP is built on IP and encapsulated in the data portion of an IP datagram. Although there are several types of ICMP messages and each has its own format, the first three fields of all ICMP messages are the same, as is denoted in Figure 4.11

![Figure 4.11: First 3 fields of Internet Control Message Protocol (ICMP).](image)

The first field (Type) defines the type of the message, and the second field (Code) provides further information about the message type. For example, if a message type is destination unreachable (Type is 3), the Code field can further explain the reason; network unreachable, host unreachable, port unreachable, or other failures.

To produce ICMP events, we use the field of event type subcode to represent the ICMP message type. Since the fields of source port and destination port in an event are not used, we can use one of them to contain the value of the code field in an ICMP
message. The length of an ICMP message can be easily calculated given the type of the message. Other fields of ICMP events can be obtained from the IP headers. ICMP events will be kept in the cache for a period of time which can be defined by users, and aggregated based on a 4-tuple (source IP, destination IP, type, code). Events will be sent to the event buffer periodically or when the cache is full.

4.1.3 Intrusive Events Generation

IntruDetector allows users to develop various intrusive simulators. In order to simulate an intrusive activity, we shall study the technique used by the intruders and examine the pattern of the traffic it generates. By doing so, we shall be able to develop a simulator equivalent to the intrusion in terms of the event stream they generate.

Various methods can be used to collect intrusion data. Well-known intrusion data can be obtained from CERT (Computer Emergency Response Team) advisories [4], and unknown intrusion data can be obtained by analyzing system vulnerabilities. To effectively simulate intrusions, we need to divide them into classes and select one or more cases from each class as the representative case. This technique is widely used in the software-testing field. It is applied here because the number of intrusions is too large to simulate all of them.

Several strategies (for example, intrusion signatures, taxonomy of system vulnerabilities) have been suggested in [48] for classifying intrusions. Users who attempt to develop simulators for a wide range of intrusions can follow one of these approaches. In this thesis, we choose several common types of intrusions as examples, and develop
one or more simulators for each type. More simulators will be developed in the future.

1. Suspicious Activities

Suspicious activities are the network traffic patterns that do not conform to the definitions of "standard" traffic. For example, in an ethernet environment, most network traffic uses the protocols of TCP, UDP, and ICMP. Any traffic of unknown protocols is suspicious and could potentially be intrusions.

2. Pre-Attack Probes

Pre-attack probes are attempts to collect information about a network and use this information for later attacks. For example, in order to penetrate into a system, an attacker may run a program to scan what types of services are running on the target. With this information, programs exploring the system vulnerability of some services may be launched to penetrate into the target.

3. Unauthorized Access Attempts

Unauthorized access attempts are attempts to access protected resources (for example, files, directories, databases, among others.) which can only be accessed by users with access privileges. Unauthorized access attempts also include attempts to gain protected access privileges. Trying to obtain root privilege on a UNIX system is an example. Many techniques can be used to acquire protected access privileges. Examples of such techniques include TFTP Get/Put and buffer overflow.

4. Denial of Service Attack
A Denial of Service Attack is an attempt to cause a system to stop providing some or all of its services to legitimate users. Examples include SYN Flood, Ping Flood, among others.

4.2 Event Buffer

To ensure that the analysis results of different analysis engines are comparable, all analysis engines should have a consistent view of the events. This is achieved by keeping events in a centralized repository until they are consumed by all analysis engines. We refer to the central repository as the event buffer. Event generators produce events and send them to the event buffer, and the analysis engines retrieve events from the event buffer and analyze them. This is actually a multiple-producer and multiple-consumer model, in which event generators are producers, and analysis engines are consumers. To improve system performance, each producer or consumer is represented by a process in the operating system. Processes can run concurrently to take advantage of the power of multiprocessor.

In the following sections, we first describe the data structures of the event buffer, then explain multiple processes control, and finally present the program interfaces for accessing the event buffer.
4.2.1 Data Structures

4.2.1.1 Event Buffer Structure

The event buffer is a shared buffer where producers store items and consumers retrieve them. Many types of data structures can be used to implement the shared buffer. For the purpose of simplicity and efficiency, we choose a First-In-First-Out (FIFO) circular queue as the data structure and use an array to implement the queue. Figure 4.12 shows a circular queue.

![Circular Queue Diagram](image)

Figure 4.12: A Circular Queue

In a general producer and consumer model, an item will be removed from the shared buffer after being consumed by a consumer. The event buffer, however, has some special features which are different from ones in a general producer and consumer model. In the event buffer,
1. An item cannot be removed from the event buffer until consumed by all consumers. This is to ensure that information sources of all analysis engines are the same, and their results can be compared on the same basis.

2. Consumers may consume items at different rates. Different analysis engines use different algorithms to analyze events. Some algorithms may be simpler and faster, while others may be more complex and slower.

Given these special features, a simple circular queue is not sufficient for implementing the event buffer. Consequently, we use a buffer consisting of three segments to implement the event buffer. Figure 4.13 shows the architecture of the buffer.

![Figure 4.13: Event Buffer Architecture](image)

4.2.1.2 Data Segment

Each item in the data segment is composed of a header and data. The data are an event record, and the header contains management information. This is illustrated in Figure 4.14.
4.2.1.3 Index Segment

Different consumers may consume events at different rates. Some may be faster than others. This requires that we maintain, for each consumer, a header pointer pointing to the first unconsumed item. At the same time, we also need to maintain
for each consumer a counter of unconsumed items. All producers share a common header pointer pointing to the first empty slot, and a tail pointer pointing to the last empty slot.

In order to prevent one consumer from going far ahead of others, a flow control mechanism needs to be implemented. For the purpose of simplicity, a consumer will be blocked after it consumes all items. This can be easily achieved by blocking a consumer when it tries to retrieve an item which has already been retrieved by that consumer before. That means the consumer's corresponding bit in the flag field has been set to zero. The blocked consumer will be awakened after a new item is generated. Figure 4.15 demonstrates the data structure of the index segment.

![Diagram of Index Segment]

**Figure 4.15: Data Structure of Index Segment**

### 4.2.1.4 Statistical Segment

The event buffer can generate some statistical information of system activities, such as how many events have been generated, how many packets or bytes of data have been produced, and so on. This information is maintained in the statistical
segment. It is denoted in Figure 4.16. Statistical information will be written to disk periodically, and system performance at different time period can be evaluated.

![Data Structure of Statistical Segment](image)

Figure 4.16: Data Structure of Statistical Segment

### 4.2.2 Process Communication and Control

To improve system performance, each producer or consumer runs independently as a process in the operating system. This brings the questions of how processes communicate with each other and how the processes are synchronized when accessing critical sections simultaneously. Interprocess communication is achieved by shared memory, and process synchronization is implemented by semaphores.

#### 4.2.2.1 Interprocess Communication (IPC)

The operating system normally provides many different mechanisms for interprocess communication. For example, in UNIX, processes can communicate with each
other by files, message passing, Remote Procedure Call (RPC), socket, shared memory, among others.

Among all IPC mechanisms, shared memory is the fastest one. Communications by all other techniques normally involve frequent data copying between the user space and the kernel. By using shared memory, processes can communicate with each other directly in the shared memory, without the involvement of the kernel. This can significantly improve system performance, especially when the communication among processes is heavy.

To achieve maximum efficiency, we choose shared memory to implement the event buffer. Shared memory is first created by a process, and then can be attached by other processes. The shared memory becomes a part of the address space of a process after it attaches to the shared memory. All the processes attaching to the shared memory can communicate with each other by exchanging information in the shared memory. Figure 4.17 demonstrates how producers and consumers communicate with each other through the event buffer implemented with shared memory.

![Diagram](Figure 4.17: Interprocess Communication By Shared Memory)
4.2.2.2 Process Synchronization

Shared memory provides an efficient method for interprocess communication. The drawback is that processes need to be synchronized to prevent the inconsistency of the shared objects. Semaphores can be used for process synchronization. A semaphore is a primitive object provided by an operating system to synchronize various processes or various threads within a process. Because the mechanism of semaphore is very reliable and widely supported in the UNIX world, it is selected for our system.

We classify semaphores into two classes, binary semaphores and counting semaphores. A binary semaphore can only have the values of 0 or 1. It is used for mutual exclusion. A counting semaphore can be initialized to any nonnegative integer value. It is used to represent the number of available resources. To synchronize event generators and analysis engines, the following semaphores are created.

- A binary semaphore for mutually exclusive accesses to the event buffer. It is initialized to 1. A process can not access the event buffer until the semaphore is acquired.

- A counting semaphore for all event generators indicates the number of free slots available in the data segment. It is initialized to the size of the array representing the data segment.

- A counting semaphore for each analysis engine indicates the number of unconsumed data items. Its initial value is 0.
High level algorithms for event generators (producers) and analysis engines (consumers) are described in Figure 4.18 and 4.19 respectively. The details of their implementations are omitted.

```
repeat
  ...
  produce an event
  wait(free slot)
  wait(event bus)
  ...
  put event to data segment
  update index segment
  update statistical segment
  ...
  signal(event bus)
  signal(event)
  ...
until false
```

Figure 4.18: Algorithm for Event Generators

4.2.3 Application Programming Interfaces

Three classes of Application Programming Interfaces (APIs) have been developed. They are used for the management of shared memory, semaphores and interprocess communication. Details of shared memory and semaphore management are hidden from users to simplify the development. Figure 4.20 denotes the hierarchy of the programming interfaces.
repeat
...
wait(event)
wait(event bus)
...
get event from data segment
update index segment
update statistical segment
...
signal(event bus)
if (a free slot is generated)
signal(free slot)
...
until false

Figure 4.19: Algorithm for Analysis Engines

4.2.3.1 Shared Memory Management

The following functions are developed for shared memory management.

- \textit{shm\_create()}. Apply for shared memory from operating system.

- \textit{shm\_delete()}. Deallocate the shared memory.
• *shm_init().* Initialize the shared memory.

• *shm_attach().* Attach to the shared memory.

• *shm_detach().* Detach from the shared memory.

### 4.2.3.2 Semaphore Management

The following functions are developed for semaphore management.

• *sem_create().* Apply semaphores from operating system.

• *sem_delete().* Release the semaphores.

• *sem_init().* Initialize the semaphores.

• *sem_open().* Open the semaphores.

• *sem_close().* Close the semaphores.

The following functions are developed for process synchronization.

• *event_bus_P().* To acquire the binary semaphore for the event buffer. If the semaphore is being occupied by another process, it will block the calling process.

• *event_bus_V().* To release the binary semaphore for the event buffer. If there are waiting processes, one of them will awaken. Which process will be awakened depends on the operating systems.
- event_generator_P(). To acquire the counting semaphore for free slots in the data segment. If the value of the semaphore is greater than 0, its value will be subtracted by one, and the calling process can continue. Otherwise, the calling process is blocked.

- event_generator_V(). To increment the value of the counting semaphore when a free slot is available. If there are waiting processes, one of them will be awakened. Which process will be awakened depends on the operating system.

- analysis_engine_P(). To acquire the counting semaphore of unconsumed events by an analysis engine. Since each analysis engine has a corresponding counting semaphore, the name of the analysis engine should be provided as a parameter. If the value of the counting semaphore is greater than zero, the value is subtracted by one and the calling process continues. Otherwise, the calling process is blocked.

- analysis_engine_V(). To inform all analysis engines when a new event is produced and sent to the event buffer. All blocked processes of analysis engines are awakened.

4.2.3.3 Process Communication

The following functions are developed for interprocess communication.

- WRITE(). This function sends an event to the event buffer. It is called by event generators when a new event is available. This function implements the semantics of a producer.
READ(). This function retrieves an event from the event buffer. It implements the semantics of a consumer. This function is called by the analysis engines.

4.3 Analysis Engine

IntruDetector can be applied to the study of different detection algorithms. In this thesis, a simplified network intrusion detection algorithm is implemented to demonstrate the applicability of Intrudetector. The algorithm is based on Network Security Monitor (NSM) [25] developed at University of California at Davis. It consists of an event assembler and two analysis algorithms, statistical and signature analysis.

4.3.1 Events Assembler

The events assembler restores TCP connections from TCP events, and organizes TCP connections to a four-layer hierarchical structure. It is illustrated in Figure 4.21.

1. Connection Layer

A TCP connection is uniquely identified by a quadruplet (source IP, destination IP, service, source port). For each TCP connection, we keep the following information:

- The number of packets transmitted over the connection.
- The number of bytes of data transmitted over the connection.

2. Service Layer
TCP connections are merged based on a triplet (source IP, destination IP, service). Service is denoted by the destination port number. After the merging, a higher layer of traffic hierarchy is obtained. It is referred to as the service layer. In this layer, we can easily learn how many packets or bytes of data have been transmitted between a pair of machines by a particular service.

3. Destination Layer

The information in the service layer can be further assembled based on the source and destination IP addresses. This layer is called the destination layer. This layer indicates how many packets or how many bytes of data have been transmitted between two machines.

4. Source Layer
The highest layer of traffic hierarchy is the source layer. It is obtained by aggregating the nodes in the destination layer based on their source IP addresses. This layer shows how many packets or bytes of data are generated by a machine.

4.3.2 Analysis Algorithms

To analyze network traffic is to search unusual traffic patterns and particular traffic patterns. The method of looking for unusual traffic patterns is called anomaly detection or statistical analysis. It is based on the Denning model [19] which assumes that intrusions will generate abnormal patterns. To look for particular traffic patterns is the approach of misuse detection or signature analysis. It assumes that each attack is associated with a particular pattern.

4.3.2.1 Statistical Analysis

In statistical analysis, current activities are compared with historical system behaviors to evaluate if they deviate significantly from the normal scope. Historical behaviors are stored in profiles created from the historical data of a system. In this thesis, the data of three weeks worth of network traffic are collected to create a historical profile representing the normal behaviors of network activities. The profile is composed of a set of records. Each record represents the historical behavior of a communication path, and consists of the following information:

- Source of the communication path.
- Destination of the communication path.
- Service of the communication path.

- Probability of the occurrence of the communication path.

- Average number of packets transmitted over the path.

- Average number of bytes of data transmitted over the path.

The profile is organized into a hierarchical structure to accelerate queries. Statistical analysis is triggered by two events:

- A new connection is established. A connection is flagged as anomalous if the probability of the occurrence of the communication path stored in historical profile is lower than a threshold.

- An alarm sounds. Statistical analysis is called periodically to check the security status of the traffic hierarchy. If the number of packets or bytes of data transmitted over a connection is unusually (for example, 5 times) higher than the average value stored in the historical profile, this connection is flagged as anomalous.

4.3.2.2 Signature Analysis

The signature analysis algorithm is called at intervals to analyze network traffic hierarchy and look for connections which violate the security rules or match any intrusion patterns. Signature analysis is also called when a new connection is established. The security rules are defined by a security officer based on the site policy. Intrusion patterns can be collected from known attacks.
4.4 Visualization Module

Visualization is widely used in many areas to help people understand complex information. We believe that visualization shall play an important role in the intrusion detection community for visualizing complicated intrusive or anomalous patterns. In this thesis, we implement a module to visualize network traffic in an interactive way. The source of the information is the traffic hierarchy created by the analysis engine. The visualization module first presents all machines which are generating network traffic in the monitored network. We call them the source machines. By clicking a source machine, all destinations with which the source machine is communicating are presented. By clicking a destination machine, the services by which the source machine speaks to the destination are displayed. By this way, we can navigate the network traffic efficiently and intuitively. The details of the visualization module are presented in Chapter 5.
Chapter 5

The Implementation of IntruDetector and Experimental Results

In Chapter 3 and Chapter 4, we described the architecture and the design of IntruDetector. In this chapter, we will present the implementation of IntruDetector. The experiments conducted with IntruDetector are also demonstrated. The chapter is organized into two sections. Section 5.1 describes the implementation of IntruDetector. Section 5.2 presents the experiments we have conducted.

5.1 Implementation

5.1.1 Programming Language Consideration

We chose C and Java as our programming languages. C is used to develop event generators, the event buffer, and the analysis engine. Java is used to implement the visualization module and the graphic user interface. Java is selected because it provides many classes for the development of graphic user interface and 2D/3D
visualization. The C programming language is chosen based on the following reasons:

- Intrudetector is developed in the UNIX environment, and many system calls are used. The system calls provided by UNIX operating system are in the form of C functions.

- The packet capture libraries provided by libpcap are C functions.

- The code generated by C compiler is very efficient. Intrudetector should be able to support real time intrusion detection, and work under a heavy load. Using C can achieve this objective.

5.1.2 User Interface Implementation

Figure 5.1 shows the user interface of Intrudetector. It consists of two sections, the menu section and the message section. The menu section consists of the menus for managing the core components. Messages will be displayed in the message section. The message section is divided into two panels, the system panel and the experiment panel. The experiment panel consists of two subpanels, one for statistical analysis and the other for signature analysis.

5.1.3 Event Generators Implementation

Event generators consist of a packet sniffer and a set of simulators. Simulators can run sequentially or concurrently. The packet sniffer runs as a daemon process. It listens to a network interface set to promiscuous mode, and generates events by parsing a bit stream of packets. The diagram is illustrated in Figure 5.2.
IntruDetector is started.

IntruDetector Analysis Engine is started.

IntruDetector is successfully started.

Figure 5.1: The Interface of IntruDetector

All events, except for emergent events (for example, TCP connection establish events), will be kept in the cache for a while. The events kept in the cache will be sent out periodically or when the cache is full.

5.1.4 Event Buffer Implementation

Two programs (called starter and stopper) are developed for starting up and shutting down the event buffer respectively. The starter creates and initializes shared memory and semaphores. It either returns gracefully or aborts when errors occur.
After the starter successfully initializes the event buffer, event generators and analysis engines can be started. To shut down the event buffer, all event generators and analysis engines should be brought down first. After all processes communicating through the event buffer are stopped, the stopper will remove the shared memory and the semaphores.
5.1.5 Analysis Engine Implementation

A detection algorithm is implemented to show the functionality of IntruDetector. It consists of an event assembler and two analysis algorithms, statistical and signature analysis.

Event assembler restores TCP sessions from events stream. TCP sessions are organized into a traffic hierarchy. The traffic hierarchy is implemented by a linked list. It is illustrated in Figure 5.3.

![Linked List for Traffic Hierarchy](image)

Figure 5.3: Linked List for Traffic Hierarchy

Statistical analysis compares the current activities with their historical behaviors, and flags a communication path as anomalous if one of the following two conditions is true.

1. The probability of the occurrence of the communication path is lower than the
threshold. The probability is stored in the historical profile.

2. The number of packets or bytes of data transmitted over the communication path is significantly (for example, 5 times) greater than the average. The average number is stored in the historical profiles.

The historical profile is created from the data of three-week worth of network traffic captured from a subnet of the Computer Science department. Figure 5.4 shows the distribution of services by which a machine communicates with others. It indicates that HTTP is used most often.

![Diagram showing distribution of services]

Figure 5.4: Distribution of Services

Signature analysis searches traffic hierarchy and flags a communication path as intrusion if it violates a security rule or matches the pattern of a known intrusion.
5.1.6 Visualization Module Implementation

A visualization module is developed for visualizing network traffic. It can show network traffic dynamically in an intuitive way based on the traffic hierarchy created by the analysis engine. Network traffic is shown in three levels.

1. The first level shows all source machines which are generating events. The events can be generated by virtual simulators or by the packet sniffer which captures and parses the network traffic produced by normal users or real simulators. Figure 5.5 shows the scenario.

2. The second level displays all destination machines with which a source machine is communicating. It will show up when a source machine in the first level is clicked. It is depicted in Figure 5.6.

3. The third level shows services through which a source machine is communicating with a destination. It will show up when a destination machine showed in the second level is clicked. A screen capture is given in Figure 5.7.

The visualization module is very helpful for testing the packet sniffer and the analysis engine. To test if the analysis engine can successfully create the hierarchy of the network traffic, we isolate the packet sniffer by running virtual simulators to generate TCP events. If the analysis engine can successfully build network traffic hierarchy from virtual events, it proves to be functional. After we know that the analysis engine works, we can test if the packet sniffer can successfully parse network packets and generate corresponding events. This can be done by simply connecting
to a machine under the monitor of the packet sniffer and monitoring if the connection shows up in the visualization module.

Figure 5.5: Machines Which Are Generating Network Traffic
Figure 5.6: Communication Paths Initiated by a Machine
Figure 5.7: Service Paths Between a Pair of Machines
5.2 Experiments

5.2.1 Objectives

We conduct a series of experiments with IntruDetector to show its capability of testing intrusion detection algorithms, and show how it is used. The experiments are designed so that the techniques of both real simulation and virtual simulation would be applied. We study the behaviors of two analysis approaches, statistical and signature analysis. These two approaches are widely used in the intrusion detection community. We will show some inherent weaknesses of each approach by our experiments.

5.2.2 Experiment Environment

We conducted the experiments in a real environment. To protect the information of network topology, all host names are pseudonymized. This is depicted by Figure 5.8. The experiment network consists of a server (S1) and several clients (C1, C2, among others). IntruDetector runs on server S1. The network is connected to the Internet. O1 represents a machine in outside.

5.2.3 Case Selection

The following testing cases are selected. Both real simulation and virtual simulation are deployed in the testing. They demonstrate the flexibility of IntruDetector.

1. An FTP session from client C1 to server S1. The session is terminated after some sensitive files are downloaded. This session is generated by real simulation.
2. A telnet session from a outside machine (O1) to server S1 at midnight. This session is generated by virtual simulation. Actually, it is very difficult to produce this session by means of the real simulation since it requires us to have access to the outside machine O1.

3. SYN-flooding denial-of-service attack on the server S1. To avoid being detected, the source of SYN messages is set to the local machine C1. The service is set to the most commonly used service, X11. This attack is simulated by virtual simulation. It is dangerous and costly to simulate the attack by real simulation even though it is possible.

5.2.4 Statistical Analysis Results

5.2.4.1 Historical Data

We collected three weeks worth of data from a subnet of the Computer Science Department. The data consist of 16922 TCP connections. Figure 5.9 shows the features of several common TCP services. We can see the number of HTTP connections
is the greatest. This is due to the nature of HTTP. When a home page is accessed, a TCP connection will be created for each object contained in the page. As most pages contain more than one object, each HTTP access may contribute several TCP connections. From the perspective of the amount of data transmitted over a connection, X11 consumes more network bandwidth than others as the average number of packets or bytes of data transmitted over a X11 connection is the greatest.

<table>
<thead>
<tr>
<th>Service</th>
<th>Number of Connections</th>
<th>Average Number of Packets</th>
<th>Average Number of Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftp</td>
<td>261</td>
<td>161</td>
<td>145,806</td>
</tr>
<tr>
<td>telnet</td>
<td>101</td>
<td>691</td>
<td>9,770</td>
</tr>
<tr>
<td>smtp</td>
<td>98</td>
<td>41</td>
<td>9,155</td>
</tr>
<tr>
<td>finger</td>
<td>6</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>http</td>
<td>10,375</td>
<td>12</td>
<td>3,332</td>
</tr>
<tr>
<td>rpc</td>
<td>786</td>
<td>8</td>
<td>134</td>
</tr>
<tr>
<td>login</td>
<td>195</td>
<td>663</td>
<td>15,053</td>
</tr>
<tr>
<td>printer</td>
<td>79</td>
<td>149</td>
<td>197,835</td>
</tr>
<tr>
<td>X11</td>
<td>169</td>
<td>26,986</td>
<td>1,089,006</td>
</tr>
</tbody>
</table>

Figure 5.9: The features of Common TCP services in Historical Profile

A subset of historical data represent the network traffic transmitted between machine CI and Server SI. The distribution of the services used by machine CI to communicate with Server SI is depicted in Figure 5.10. We can see that X11 is used most often. FTP is also frequently used.
5.2.4.2 Analysis Results

- Case 1. This session is not flagged as suspicious since the probability of the occurrence of the session is higher than the predefined minimum probability. However, this session is actually very suspicious. It downloaded sensitive system files, and could use them to crack user passwords. Since the session is initiated from a local machine, it could be done by an internal user. It is also possible that machine $C1$ has been cracked and used as the basis to launch attacks against others. The failure to identify this suspicious activity shows the weakness of statistical analysis for detecting misuse.

- Case 2. This session is flagged as anomalous because this session did not occur before. Although this session is suspicious, it is also possible that the session is initiated by an internal user who is on out-of-town business. Flagging this session as intrusive indicates that statistical analysis may trigger false alarms.
This is due to the immaturity of the detection algorithm. If the algorithm can adapt to the changing of user behaviors, false alarms may be reduced.

- **Case 3.** This session is not flagged as suspicious since HTTP is the protocol used most often in the experimental network. Figure 5.7 shows that 63 percent of connections are HTTP connections. The inability to identify SYN-flooding attack is due to the simplicity of the history profile. If the profile contains more information, such as the intensity of connections, it would be possible to detect denial of service attacks like SYN-flooding attack. In this attack, the technique of IP spoofing is used. The real source address of the intrusion is set to the local address to spoof the detector. This can be detected by the router sitting between the local network and the Internet. Any packets coming from outside with inside addresses should be dropped by the router since they explicitly violate the principle of network operation.

### 5.2.5 Signature Analysis Results

#### 5.2.5.1 Security Rules and Signatures

Security rules are site dependent. They are defined by a security officer according to the organization policies of how the computing resources should be used. In our experiments, we define the following security rules. Any violation of these rules is deemed as intrusive.

- Access to server $S1$ is restricted to the inside network.

- The maximum number of concurrent connections to $S1$ is 1000.
5.2.5.2 Analysis Results

- Case 1. The FTP session from \( C1 \) to \( S1 \) is not flagged as an intrusion since it does not violate any security rules. Although downloading sensitive files is abnormal, it cannot be detected. This is due to our implementation, in which we only utilize the information contained in the packet headers, but do not analyze any data. It is impossible to know which files have been accessed without parsing the data of packets.

- Case 2. The telnet session is flagged as intrusive as it explicitly violates rule 1.

- Case 3. Some of TCP SYN events are flagged as intrusions. Signature analysis only checks if the security rules are violated. When we begin to simulate SYN-flooding attack, the total number of current connections is less than 1000. When the limit is reached, all the new connections are flagged as intrusive. Although some malicious connections can be found, many normal activities are also flagged as intrusions. There are many false alarms. To effectively detect this type of attacks, a simple rule is insufficient. We need to build a signature database and apply sophisticated techniques (for instance, expert systems) to match the current activities with the known intrusive patterns.
5.2.6 Summary of Experiments

We have conducted several experiments with Intruder. The techniques of hybrid simulation are used in our experiments. Real simulation is used to simulate nondestructive intrusions, and virtual simulation is used to simulate destructive intrusions or some activities which are not suitable for real simulation. This demonstrates the advantages of hybrid simulation, and show the capability of Intruder for testing network intrusion detection algorithms.
Chapter 6

Conclusions

Intrusion Detection Systems (IDSs) play an important role in protecting information infrastructure. Most of the current detection algorithms are not sufficiently mature and require extensive testing and further improvement. In this study, we develop a software platform, named IntruDetector, which can be used to test network intrusion detection algorithms in a real environment with a wide range of intrusive cases without any danger of compromising normal system operations.

6.1 Summary of IntruDetector

IntruDetector consists of four components: the event generators, the event buffer, the analysis engine, and the visualization module. The event generators can generate events representing both normal and intrusive network activities. Normal events are produced by the packet sniffer from a stream of packets transmitted over the monitored network. Intrusive events can be generated by simulation. The event buffer is a central repository where event generators store events which are retrieved
by analysis engines. It provides a mechanism to ensure that all analysis engines have a consistent view of events and their analysis results can be compared on the same basis. The analysis engine is developed to test IntruDetector and show its applicability.

The analysis engine uses statistical and signature approaches to analyze events and detect any anomalous or intrusive activities. The visualization module presents the results of the analysis engine to a graphic user interface. It can also visualize network traffic in an interactive way. The visualization module helps us tremendously in the development and the testing of the platform. The system has the following features:

- **Realism.** Testing of intrusion detection algorithms can be conducted in a real environment and is kept from the danger of compromising system normal operations.

- **Applicability.** By using the technique of hybrid simulation, a variety of types of intrusion can be simulated.

- **High performance.** Event generators and analysis engines run concurrently, taking advantage of the power of a multiprocessor system. The performance of interprocess communication is significantly improved by using shared memory since kernel involvement is not necessary in the interprocess communication.

- **Extensibility.** Simulators and analysis algorithms can be easily added by users. The templates of simulator and analysis algorithm are also provided. The system can also be extended to support other types of protocols (for example, IPX).
• Flexibility. Simulators can be run at any time without interrupting the system. They can run sequentially or concurrently since each simulator is an independent process.

• Standardization. The design of the platform conforms to the current ongoing standards, CIDF. Event generators are equivalent in function to CIDF E-boxes. The event buffer is similar to CIDF D-boxes. Analysis engine can be regarded as the CIDF A-boxes.

6.2 Main Contributions

The main contributions of this thesis are as follows:

• A software platform, IntruDetector, is designed and implemented. It can be used as an environment to test network intrusion detection algorithms.

• The technique of hybrid simulation is utilized to generate normal and intrusive events. It allows the testing to be performed in a real environment with a wide range of intrusive activities, but without any danger of compromising normal system operations.

6.3 Future Directions

The core of an intrusion detection system is its detection algorithms. IntruDetector allows intrusion detection algorithms to be studied in a real environment and tested with a wide range of intrusions. Future work should focus on developing new
intrusive simulators and detection algorithms. A template of simulator is provided with the system. Researchers can develop new simulators by duplicating the template and adding into it any new activities. To facilitate the development of detection algorithms, a dummy analysis module is provided. It retrieves events from the event buffer, but does not do any analysis. Researchers may insert any new analysis algorithms into the module. By thoroughly testing detection algorithms in a real environment and with a wide range of intrusive activities, the detection algorithms can be improved and become more mature. With mature detection algorithms, intrusion detection systems can be applied in a real environment to effectively deter computer intrusions. The current limitation is that IntruDetector is designed based on Ethernet and UNIX environment, and it does not support other types of networks (for example, FDDI and ATM). It will also require further development to migrate IntruDetector from UNIX to Windows.
Bibliography


92


Appendix A

ACRONYMS

AAAI: American Association for Artificial Intelligence
ACK: Acknowledgment
ACM: Association for Computing Machinery
AI: Artificial Intelligence
API: Application Programming Interface
ATM: Asynchronous Transfer Mode
BPF: BSD Packet Filter
BSD: Berkeley Software Distribution
CERT: Computer Emergency Response Team
CIDF: Common Intrusion Detection Framework
CISL: Common Intrusion Specification Language
COAST: Computer Operations, Audit, and Security Technology
CS: Computer Science
DARPA: Defense Advanced Research Projects Agency
DIDS: Distributed Intrusion Detection System
DLPI: Data Link Provider Interface
EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances
FDDI: Fiber Distributed Data Interface
FIFO: First In First Out
FTP: File Transfer Protocol
GIDO: Generalized Intrusion Detection Object
GrIDS: A Graph Based Intrusion Detection System
HTTP: Hypertext Transfer Protocol
IBM: International Business Machines Corporation
ICMP: Internet Control Message Protocol
IDES: Intrusion Detection Expert System
IDIOT: Intrusion Detection In Our Time
IDS: Intrusion Detection System
IDWG: Intrusion Detection Working Group
IEEE: Transactions on Software Engineering
IESG: Instructional & Electronics Support Group
IETF: Internet Engineering Task Force
IP: Internet Protocol
IPC: Interprocess Communication Communication
IPX: Internetwork Packet eXchange
ISO: International Organization for Standardization
LAN: Local Area Network
MIT: Massachusetts Institute of Technology
NADIR: Network Anomaly Detection and Intrusion Reporter
NIDES: Next-generation Intrusion-Detection Expert System
NSM: Network Security Monitor
OS: Operating System
OSI: Open System Interconnection
P-BEST: Production-Based Expert System Toolset
RAID: Recent Advance in Intrusion Detection
RFC: Request for Comments
RPC: Remote Procedure Call
SLIP: Serial Line Internet Protocol
SRI: Stanford Research Institute
SSO: System Security Officer
SVR4: System V Release 4
SYN: Synchronization
TCP: Transmission Control Protocol
TFTP: Trivial File Transfer Protocol
UC: University of California
UDP: User Datagram Protocol