DESIGN AND IMPLEMENTATION OF SNORT IDS RULES MANAGEMENT

CHITTHAPORN SAE-NGOW

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF SCIENCE
(COMPUTER SCIENCE)
FACULTY OF GRADUATE STUDIES
MAHIDOL UNIVERSITY
2007

Thesis
Entitled

DESIGN AND IMPLEMENTATION OF SNORT IDS RULES MANAGEMENT

……...
Miss Chitthaporn Sae-Ngow,
Candidate

……...
Lect. Sudsanguan Ngamsuriyaroj, Ph.D.
Major-Advisor

……...
Asst.Prof. Vasaka Visoottiviseth, Ph.D.
Co-Advisor

……...
Lect. Songsri Tangsripairoj, Ph.D.
Co-Advisor

……..
Prof. Banchong Mahaisavariya, M.D.
Dean
Faculty of Graduate Studies

……...
Assoc. Prof. Supachai Tangwongsan, Ph.D.
Chair
Master of Science Programme
In Computer Science
Faculty of Science

Thesis
Entitled

DESIGN AND IMPLEMENTATION OF SNORT IDS RULES MANAGEMENT

was submitted to the Faculty of Graduate Studies, Mahidol University
for the degree of Master of Science (Computer Science)
on
October 5, 2007

……...
Miss Chitthaporn Sae-Ngow,
Candidate

……...
Lect. Panomporn Suvannapattana, Ph.D.
Chair

……...
Lect. Sudsanguan Ngamsuriyaroj, Ph.D.
Member

……...
Asst.Prof. Vasaka Visoottiviseth, Ph.D.
Member

……...
Lect. Songsri Tangsripairoj, Ph.D.
Member

……..
Prof. Banchong Mahaisavariya, M.D.
Dean
Faculty of Graduate Studies
Mahidol University

……...
Prof. Skorn Mongkolsuk, Ph.D.
Dean
Faculty of Science
Mahidol University

ACKNOWLEDGEMENT

The success of this thesis is dedicated to my advisor, Dr. Sudsanguan Ngamsuriyaroj, who has been giving me invaluable guidance, advice and continuous support during my research on this thesis.

I am also grateful to Dr. Vasaka Visoottiviseth, Dr. Songsri Tangsripairoj and Dr. Panomporn Suvannapattana for their suggestions and constructive feedback during my thesis presentation.

My appreciation also goes to Assoc. Prof. Supachai Tangwongsan, Assoc.Prof. Jarernsri L. Mitrpanont and Lect. Pagaporn Pengsart who I will never forget their kindness during my work at Mahidol University Computing Center.

In addition, I wish to thank all my colleagues and graduate friends for their friendships. Many thanks also go to the Department of Computer Science for supporting computers and equipments used in my experiments.

Finally, my deepest appreciation and gratitude go to my parents, my sister and my brother who have given me not only educational support but also morality, endless love and warm encouragement.

Chitthaporn Sae-Ngow

DESIGN AND IMPLEMENTATION OF SNORT IDS RULES MANAGEMENT

CHITTRAPORN SAE-NGOW 4537073 SCCS/M

M.Sc. (COMPUTER SCIENCE)

THESIS ADVISORS: SUDSANGUAN NGAMSURIYAROJ, Ph.D., SONGSRI TANGSRIPAIROJ, Ph.D., VASAKA VISOOTTIVISETH, Ph.D.

ABSTRACT

Mahidol University's network or MUC-Net has deployed several security mechanisms including firewalls, anti-virus, and an intrusion detection system or IDS. But, the distributed structure of Mahidol University's campuses makes it difficult to detect and prevent intrusions from inside the network. One essential idea is to install multiple IDS systems at several locations so that individual IDS would detect any malicious attempts and send alerts to the central administrators in-charge of the time.

SNORT is the most popular open source IDS based on a set of signatures or rules for detecting intrusions. Rule updates are primarily done manually by the system administrator. But, for distributed nature of MUC-Net, the problem of updating rules to ensure their consistency must be addressed to reduce risks and damages from attacking. In addition, having every rule active would degrade the performance of SNORT machines. Hence, the usage of SNORT rules at all locations must be monitored.

In this thesis, we designed and implemented rules management for a distributed SNORT IDS that consisted of IDS sensors installed at several locations, and the IDS manager who automatically updates SNORT's rules at every IDS sensor when they are downloaded from snort.org. IDS sensors not only keep track of rules usage at their machines to minimize the number of active rules, but also ensure rules consistency among them. The system also analyzes rules usage of individual IDS. Thus, the IDS's performance would be enhanced. We install SNORT IDS sensors at four computer labs, and alert logs for one month periods are collected and analyzed. We found that only a few rules are actively used, and the number of false alarms is high for SNMP-type since most network equipments use it for monitoring.

KEY WORDS: INTRUSION DETECTION SYSTEM/ RULES MANAGEMENT / RULES CONSISTENCY/ FALSE ALARMS

70 pp.

การออกแบบและพัฒนา การบริหารจัดการเงื่อนไขของการตรวจสอบการบุกรุกของระบบด้วยระบบการตรวจสอบผู้บุกรุก SNORT (DESIGN AND IMPLEMENTATION OF SNORT IDS RULES MANAGEMENT)

จิตราภรณ์ แซ่โง้ว 4537073 SCCS/M

วท.ม. (วิทยาการคอมพิวเตอร์)

คณะกรรมการควบคุมวิทยานิพนธ์: สุดสงวน งามสุริยโรจน์, Ph.D., ทรงศรี ตั้งศรีไพโรจน์, Ph.D., วัสกา วิสุทธิวิเศษ, Ph.D.

บทคัดย่อ

เครือข่ายมหาวิทยาลัยมหิดลหรือ MUC-Net มีระบบรักษาความปลอดภัยหลายชนิดได้แก่ ระบบ Firewall ระบบ Anti-virus และระบบตรวจจับผู้บุกรุกหรือ IDS แต่ลักษณะโครงสร้างแบบกระจายของมหาวิทยาลัยมหิดลนั้นยากแก่การตรวจจับและป้องกันผู้บุกรุกจากภายใน ดังนั้นจึงเสนอวิธีการติดตั้งระบบ Multiple IDS โดยแต่ละ IDS สามารถตรวจจับการบุกรุก และส่ง Alert เตือนไปยังผู้ดูแลระบบในส่วนกลางได้ทันเวลา

Snort เป็น Open-Source เกี่ยวกับระบบตรวจจับผู้บุกรุกที่ได้รับความนิยมมากที่สุด โดยใช้หลักการอ้างอิง Signature หรือ Rules ในการตรวจจับผู้บุกรุก ส่วนมากแล้วผู้ดูแลระบบจะเป็นผู้ Update Rules เอง สำหรับเครือข่าย MUC-Net ที่มีลักษณะแบบกระจายนั้น ปัญหาในการ Update Rules ให้มีความสอดคล้องกันจึงเป็นส่วนสำคัญ เพื่อที่จะลดความเสี่ยงและความเสียหายจากการบุกรุก นอกจากนั้น หากใช้งานทุก Rules ที่มีทั้งหมดอาจส่งผลกับประสิทธิภาพในการทำงานของ Snort อีกด้วย ดังนั้นจึงควรที่จะติดตามการใช้งาน Rules ของ Snort ในทุกแห่งที่ติดตั้ง Snort ด้วย

ในงานวิจัยนี้ เราได้ออกแบบและพัฒนาระบบการบริหารจัดการ Rules สำหรับระบบ Snort IDS แบบกระจาย ที่ประกอบด้วย IDS Sensors ทำการติดตั้งไว้แต่ละแห่ง และมี IDS Manager ทำหน้าที่ Update Snort Rules ให้กับ IDS Sensors แบบอัตโนมัติ เมื่อได้ Download จาก snort.org และ IDS Sensors ไม่เพียงทำการตรวจดูการใช้งาน Rules ของเครื่อง เพื่อลดจำนวนการใช้ Rules เท่านั้น แต่ยังคงความสอดคล้องของ Rules ทั้งระบบอีกด้วย ซึ่งระบบทำการวิเคราะห์การใช้งาน Rules แต่ละเครื่อง ดังนั้นและประสิทธิภาพจะเพิ่มมากขึ้น เราได้ทำการติดตั้ง Snort IDS ที่ Lab จำนวน 4 เครื่อง และทำการจัดเก็บ Alert Logs และวิเคราะห์ เป็นเวลา 1 เดือน ทำให้พบว่ามี Rules เพียงเล็กน้อยที่ถูใช้งาน และพบ False Alarms ประเภท SNMP จำนวนมาก จากการเฝ้าสังเกตการทำงานของเครือข่าย

70 หน้า

CONTENTS

CONTENTS (Cont.)

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF FIGURES (Cont.)

Page

CHAPTER 1
INTRODUCTION

The Internet now becomes ubiquitous either on wired, wireless or mobile networks as a large number of services could be easily delivered through the Internet. The Internet, however, has both advantages and disadvantages because, while everyone can freely and happily access abundance of data and services, some groups of people use the Internet with ill intentions. As a result, the problems of security are always increased in proportion to higher communication speed.

Currently, all kinds of threats such as virus, worm, and denial-of-service attacks are happening every day. In effect, detection and prevention mechanisms such as antivirus software and intrusion detection systems are installed to prevent such threats. Without effective protection, the performance of computer and network systems would be directly impacted, and may not give services as usual. Thus, research and commercial communities have put lots of efforts to find specific patterns of attacks so that predefined rules or signatures could be applied when a real attack comes. However, new patterns of threats are always invented such that antivirus or intrusion detection software could not detect or even prevent them.

CERT, standing for Computer Emergency Response Team, is responsible for safeguarding computers and networks as it will be informed of any threats happening in this digital world. Thus, it will collect threat information and post solutions to detect and prevent such threat. Based on monthly CERT reports, there have been more and more attacks which result in sudden interruption of an organization’s networks and services. Sometimes, they cause lots of damages to essential data. Several threats such as virus, data robbery, spyware as well as intrusions on networks had caused delays or interruptions to user services. Although recent security technologies such as firewall, anti-virus, anti-spyware and encryption could help prevent such threats efficiently, they are not always sufficient to protect the networks within an organization.

As a result, a system for detecting security violation such as intrusion detection system (IDS) and intrusion prevention system (IPS) has been installed in most networks. In addition, open source IDS systems such as Snort [11, 13, and 23] and Bro [2] are freely available and widely used. Both systems typically have a small database for storing intrusion signatures used to verify incoming traffic. However, it is crucial to have an up-to-date signature database to handle new and recent intrusions since a variety of intrusions happened every day.

Mahidol University network called MUC-Net interconnects faculties and institutions located across four campuses: Phayathai, Salaya, Siriraj and Kanchanaburi as illustrated in Figure 1.1. To detect and prevent intrusions from outside and inside MUC-Net, IDS machines are installed at several places within the network so that whenever there is an attack, the corresponding signature will detect the attack at one of the IDS machines, and appropriate alert logs will be produced. The log should be used to alert other IDS servers so that such recent intrusions could be detected and prevented across campuses as soon as possible.

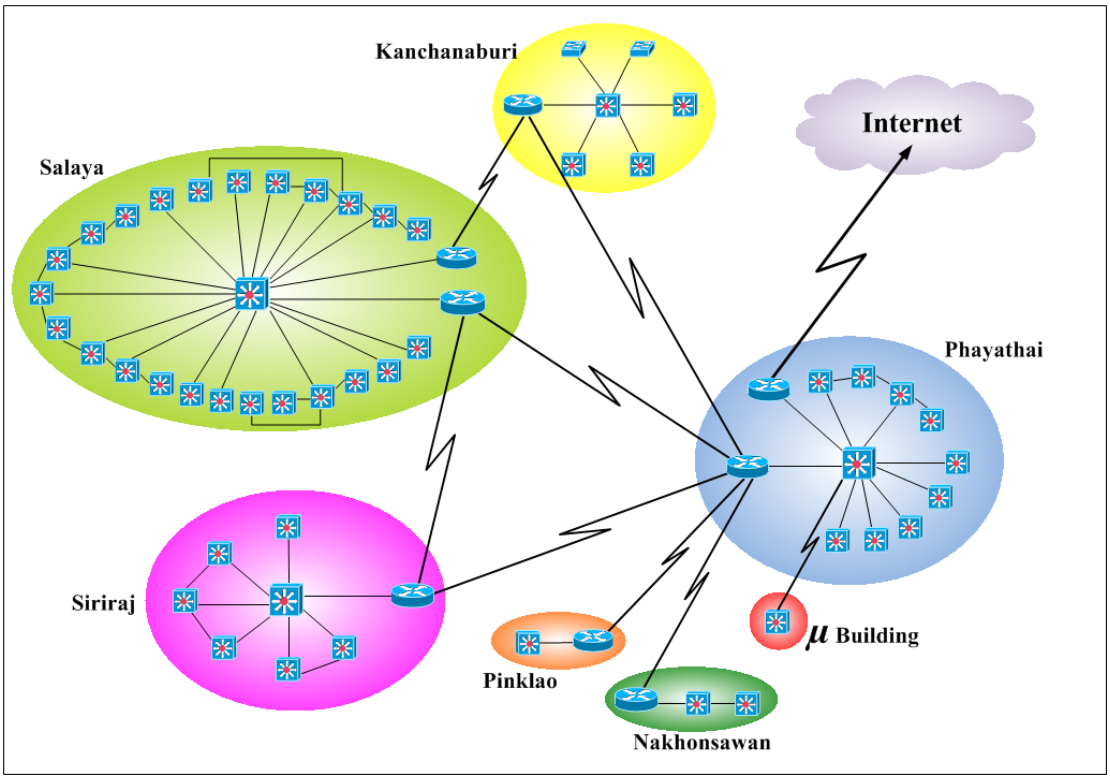

Figure 1.1: Overview of MUC-Net

This research aims to monitor the usage of signature rules on all IDS machines installed across campuses of Mahidol University in order to minimize the number of rules deployed at any moment such that the IDS performance is still efficient. We also focus on managing the updates of signature rules on those IDS servers. In addition, the alert logs recorded locally at each IDS server will be sent to the centralized management node so that they can be analyzed for rules usage and for finding any existing false alarms. The proposed system will record patterns of intrusions via alert logs, analyze which rules are used to create such logs, and report such logs to the centralized management node. This research will develop a mechanism to update intrusion signatures on several IDS servers that would fit a distributed nature of Mahidol University network.

1.1 Problem Statement

Most IDS systems rely on the signatures of intrusion patterns that would need to be frequently updated so that those IDS would be strongly resistant to recent attacks. However, most signatures are manually updated by the system administrator. In a distributed network like Mahidol University network where several IDS servers are installed, updating such signatures manually would be complicated and not as efficient as a small network.

The efficiency of Snort, one of the widely used open source IDS, depends on such signature updates as well. In addition, updating signatures on several Snort machines may cause some inconsistency, and may increase a high risk for intrusions to the whole campus.

Thus, in this thesis, we design a distributed IDS management system that would help us efficiently and accurately update the most recent intrusions signatures from one IDS server to other IDS servers distributed across campuses. Snort will be installed and signatures residing locally on each IDS will be periodically updated. Thus, the signature transfer among those IDS servers must be performed.

1.2 Research Objectives

1. To develop a distributed IDS system based on Snort IDS.
2. To monitor which rules are active at which IDS sensor site.
3. To update intrusion signatures installed locally at various sites and make sure that those signatures are always up-to-date and consistent so that intrusions can be prevented and potential risks incurred could also be minimized.
4. To find a way to reduce any false alarms so that the number of alert logs would be reduced since some network monitoring system may cause such alarms enormously.

1.3 Scope of the Research

This research focuses on the following scopes.

1. Snort used as our IDS are installed as a sensor at four different sites of Mahidol University.
2. Snort alert logs are collected at all local sites, and they are preliminary processed before sending the summary results to a centralized management server.
3. A centralized management server is responsible for receiving alerts from each IDS sensor, processing those alerts, and finding false alarms.
4. A centralized management server is responsible for updating signatures located at all IDS sensor sites.
5. A centralized management server is responsible for reporting rules usage at all IDS sensor sites.

1.4 Chapter Organization

This research is organized into five chapters described as follows.

Chapter 1: Introduction

This chapter presents the motivation and the objectives of the research.

Chapter 2: Background

This chapter describes the research background on several IDS systems developed as open source software. They include Snort and Bro. Rules or signatures structures of Snort are also explained.

Chapter 3: Related Work

This chapter describes research work that is related to our work.

Chapter 4: Proposed Work

This chapter presents the proposed work. The model, the process and the designed database used to develop this research are described in detail.

Chapter 5: Implementation and Experiments

This chapter explains the implementation and the experiments conducted in this work.

Chapter 6: Discussion and Conclusion

This chapter discusses the experimental results, makes the conclusions, and gives suggestions for future work.

CHAPTER 2
BACKGROUND

This chapter gives some background on intrusion detection systems and some full-fledged developed IDS such as Snort[11, 13, 23], Bro[2] and Prelude[24]. We also explain the system called Honeynet[3, 22] that could learn attacking patterns, and a standard message exchange for IDS data called Intrusion Detection Message Exchange Format (IDMEF)[14].

2.1 Intrusion Detection System (IDS)

IDS could be classified into three groups as follows.

- Host-based IDS is an IDS installed at a machine to detect intrusions at the machine itself. The IDS analyzes events happened on the machine using the data derived from system service requests, file-system changes, and memory allocation.
- Network-based IDS analyzes the incoming traffic from the network to find out whether the traffic contains any intrusions. It also acts as a gateway that allows or denies packets for passing thru from one network to another. Thus, the performance bottleneck may arise. Furthermore, if the incoming traffic was encrypted, the analysis would not be possible.
- Hybrid IDS is the combination of host-based and network-based IDS. Typically, it has a centralized system to handle the system performance while allowing the data from the sensor nodes to be delivered to the center node. The performance of a hybrid IDS greatly depends on the number of resources available and used.

Due to different threats of recent intrusions, open source IDS software such as Snort, Bro, and Prelude has been developed. Snort is the most widely used IDS and has also been extensively studied. However, like most IDSes, Snort is mainly categorized as a signature detector and unable to detect new or unknown attacking

patterns. Thus, Snort heavily relies on signatures stored in the database. But, as more intrusions happened daily, a number of signatures or rules in Snort is dramatically increased and is now over 10,000.

2.1.1 Snort[11, 13, 23]

Snort is an open source network IDS that can efficiently detect malicious traffic on a network. It uses the signature detection method to examine packet contents whether they are matched with one of the predefined rules. If it is a match, an alert log is created and sent to the system administrator. The main processes of Snort are described in the diagram of Figure 2.1, and consist of three major components as explained below.

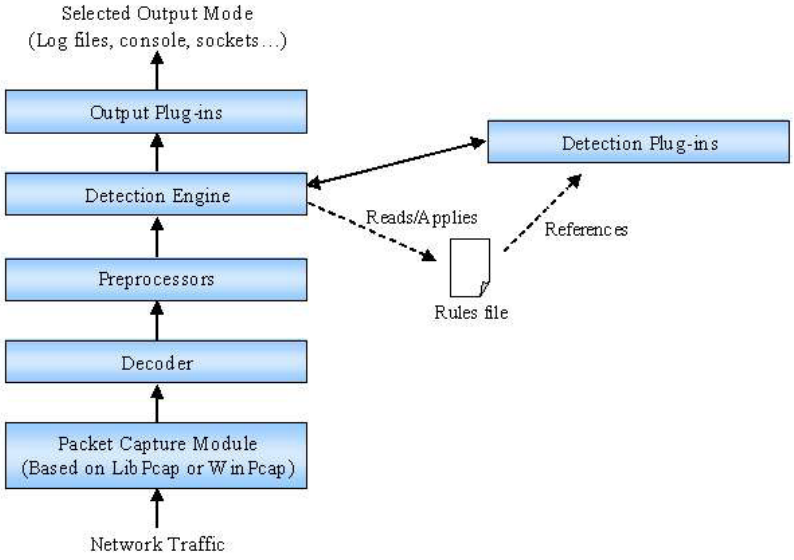

Figure 2.1: Snort Block Diagram [11]

1. Packet Decoder is responsible for capturing packets, decoding and analyzing input packets using *libpcap* from the data link layer to the application layer of the protocol stack. All packets will be preprocessed and forwarded to the detection engine.

2. Detection Engine is the most significant part. It serves to analyze and verify packets to find whether they are positively matched with active Snort rules as the signature detection is applied. Snort rules are stored as a set of text files, and they could be added or updated as needed. Since all packets must be examined thru the set

of predefined rules, the performance of intrusion detection could be slow down if the number of rules is high. For effective packet classification, Snort rules are designed as a structure of the two-dimension linked list displayed in Figure 2.2 and they are composed of two parts as follows.

1. Chain Headers tell the type of the packet's network protocol such as TCP, UDP and ICMP. They also tell IP source and destination as well as source and destination ports.
2. Chain Option tells the rule property and the details of each rule for attack identification.

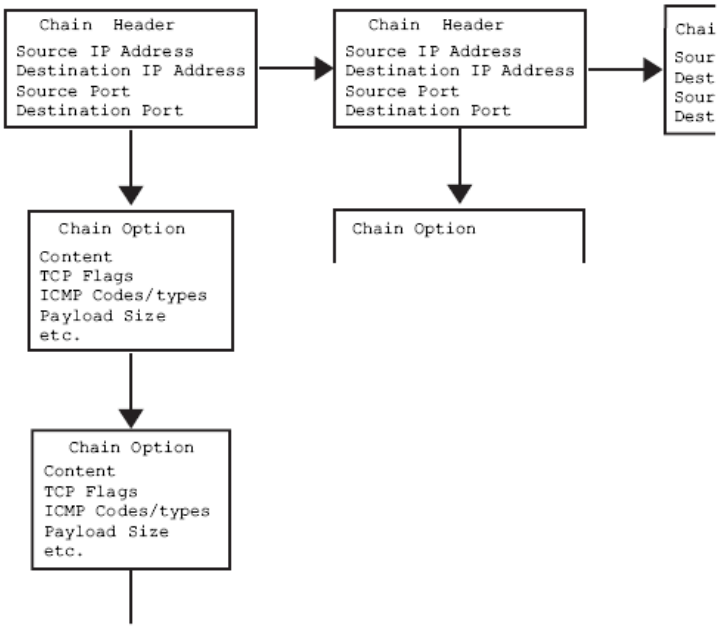

Figure 2.2: Snort's Rule Structure

If packets are positively matched with one signature, Snort could take three actions below.

- *Pass Rule*: Snort drops packets.
- *Log Rule*: Snort stores the data of input packets in a log file.
- *Alert Rule*: Snort stores the data of input packets in a log file, and also alerts the system administrator.

Rules in Snort has the predefined pattern as shown below.

<Action Rule> <Protocol> <In bound> <Out bound> → <Destination Network/netmask> <Port> (< Destination Network> <port> (<Content : " xxxxx " >)

The followings are examples of defined Snort rules.

```
Pass  tcp  any  any → 192.168.10.0/24 (Content: "|E8C0 FFFF FF| /bin/sh";)
Alert tcp  any  any → 10.1.1.0/24  6000:6010 (msg: "X traffic";)
Log   tcp  any  !10.1.1.0/24  → 10.1.1.0/24  6000:6010 (msg: "X traffic";)
```

3. Logging/Alerting could be selected through the command line. The logging has three options given as follows.

- Log packets are decoded so that the system can analyze them once again.
- Log packets are in the format that the system administrator can read.
- Log packets are in the *tcpdump* binary format.

The alert option serves to send a warning statement to the system syslog or send alerts to the system administrator. Snort does not recommend using both parts simultaneously since they will significantly slow down the system performance and reduce the efficiency drastically.

The number of rules on Snort 2.6 is currently 8,255 rules. They are classified into 46 categories as shown in Table 2.1. Obviously, the attack category with the highest number of rules is NetBIOS while rules on web categories are high as well. Hence, such rules reflect high possibility of such threats.

2.1.2 Bro[2]

Bro is an open-source network IDS that monitors network traffic and looks for suspicious activities. Bro detects intrusions by first parsing network traffic to extract application-level semantics and then executing event-oriented analyzers that compare the activity with patterns deemed troublesome. Its analysis includes the detection of specific attacks and unusual activities such as certain hosts attempted connecting to certain services, or patterns of failed connection attempts. Bro's structure is shown in Figure 2.3 and its three components are described as follows.

1. Libpcap is the packet capture library used by *tcpdump*. It filters the incoming packet stream from the network to extract the required packets such as listening for port finger, port ftp, tcp port 113 (Ident), port telnet, port login, or port 111 (Portmapper). Moreover, the Libpcap can capture packets with the SYN, FIN, or RST control bits set.

Rule Number	Rule Name	Count
1	attack-responses	16
2	backdoor	708
3	bad-traffic	6
4	chat	33
5	ddos	30
6	dns	21
7	dos	14
8	exploit	152
9	finger	13
10	ftp	78
11	icmp	22
12	icmp-info	93
13	imap	56
14	info	6
15	misc	55
16	multimedia	10
17	mysql	14
18	netbios	3,570
19	nntp	12
20	oracle	298
21	other-ids	3
22	p2p	23
23	policy	48
24	pop2	4
25	pop3	35
26	porn	21
27	rpc	147
28	rservices	13
29	shellcode	31
30	smtp	85
31	snmp	16
32	specific-threats	5
33	spyware-put	734
34	scan	12
35	sql	83
36	telnet	21
37	tftp	15
38	virus	6
39	web-cgi	355
40	web-client	704
41	web-coldfusion	44
42	web-frontpage	38
43	web-iis	95
44	web-misc	368
45	web-php	140
46	x11	2

Table 2.1: Number of Snort Rules in 46 Categories

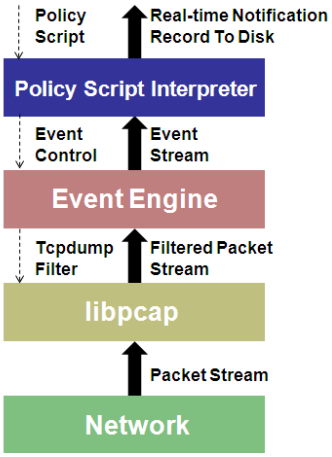

Figure 2.3: Structure of the Bro system

2. Event Engine The filtered packet stream from the Libpcap is handed over to the Event Engine which performs several integrity checks to assure that the packet headers are well formed. It looks up the connection state associated with the tuple of the two IP addresses and the two TCP or UDP port numbers. It then dispatches the packet to a handler for the corresponding connection.

For each TCP packet, the connection handler verifies that the entire TCP header is present and validates the TCP checksum. If successful, it tests whether the TCP header includes any of the SYN/FIN/RST control flags and adjusts the connection's state accordingly. Different changes in the connection's state generate different events.

3. Policy Script Interpreter receives the events generated by the Event Engine. It executes scripts to generate events like logging real-time notifications, recording data to disk and modifying internal states. Adding new functionality to Bro consists of adding a new protocol analyzer to the event engine and writing new events handlers in the interpreter.

2.1.3 Prelude[24]

Prelude is a hybrid IDS having the characteristics of both host-based IDS and network-based IDS. Prelude merges the two characteristics by developing the data format called Intrusion Detection Message Exchange Format (IDMEF) [14] for data to be delivered between IDS sensor nodes. IDMEF is expected to be the standard format of data delivery for IDS in the future.

Prelude consists of five modules and its architecture is illustrated in Figure 2.3.

1. Sensors serve to detect specific events on a particular segment of the network. If a designated event is found, they will send an alerting signal to the Prelude manager.

2. Prelude Manager is the centralized node for processing intrusion information received from sensors. If there is more than one manager in the Prelude system, all alert messages will be forwarded to other managers as well as to counter measure agents.

3. Counter Measure Agents serve to make a decision when abnormal events happened in the system and alerts are reported.

4. Frontend is part of a statistical report when some abnormal events arise or there are attacks. Frontend may reside on the same computer as a Prelude manager.

5. LibPrelude is the library used for communicating among different components so that they can work together via the IDMEF format.

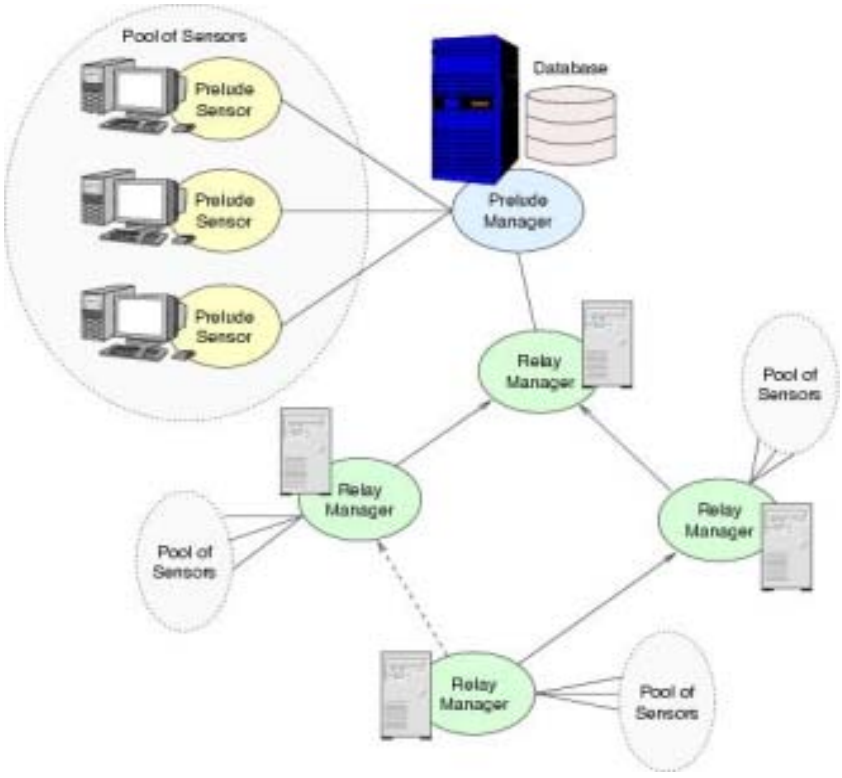

Figure 2.4: Prelude Architecture

2.1.4 Honeynet[3, 22]

Honeynet is evolved from the Honeypot project that learns intrusion patterns. But, Honeynet aims to learn the patterns of attacks and also prevent intrusions. Honeynet uses Snort-inline as the main assistant, and consists of three parts as follows**.**

1. Data Control serves to analyze incoming packets to find whether they are intruding packets. It also coordinates with Snort-inline in order to reduce the amount of intrusion traffic by limiting connections coming into the system.

2. Data Capture serves to monitor and record intruding data. Levels of data capture include network activity, application activity and system activity.

3. Data Analysis aims to collect and learn intrusion patterns. The data analysis is designed to operate via a web browser as illustrated in Figure 2.4 below.

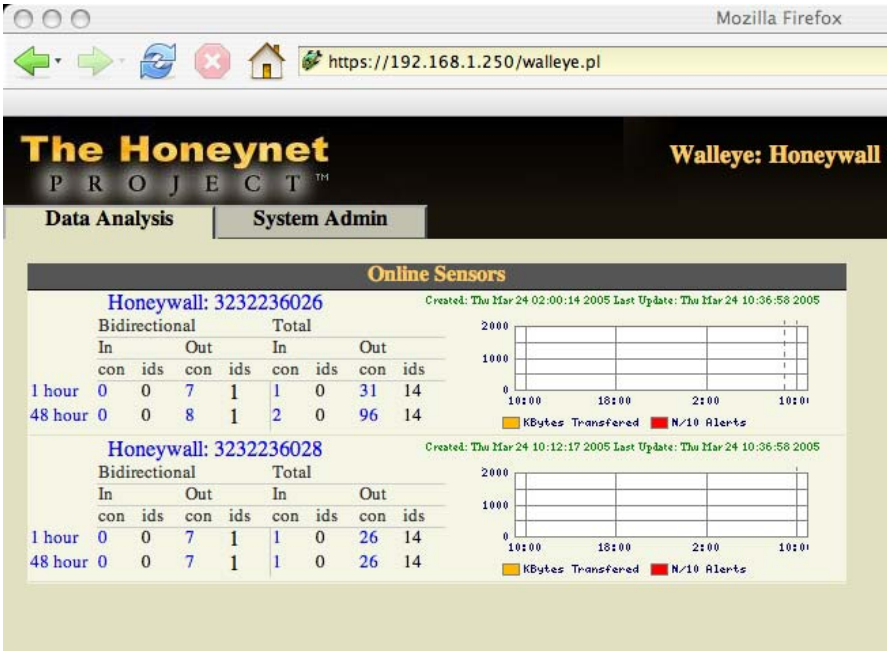

Figure 2.5: Honeynet Data Analysis

Honeynet mainly focuses on data control. It is placed between network devices and computers that need protection from intruders. The connection in this manner is called a bridged or transparent mode. Additionally, Honeynet enhances its detection efficiency using the concept of firewall in *iptables* which arranges packets in a queue before sending them to Snort-inline for verification. Honeynet is operated in three modes described as follows.

- *Connection limiting mode*: The number of incoming packets is limited by a specified threshold. If the number is more than the threshold defined, the system will not accept those packets, and drop them.
- *Snort-inline drop mode*: When the incoming packets contain information matched with the rules, they will be dropped.
- *Snort-inline replace mode*: It alters dubious packets but does not drop them. Particularly, those packets may perform erroneously when they reached the destination such as changes of bits in packets.

2.2 Intrusion Detection Message Exchange Format (IDMEF)[14]

Intrusion detection message exchange format (IDMEF) is developed from one group of the Internet Engineering Task Force (IETF) called Intrusion Detection Working Group (IDWG). Its objective is to establish the standard for data delivery in IDS systems.

IDMEF defines a data mode regarding “Alert” and other data expressed in the extensible markup language (XML) format. It consists of two components depicted in Figure 2.5 and described as follows.

- **Alert part** contains intrusion information such as intrusion time, source IP, target IP, protocol, and port.
- **Heartbeat part** tells us how often the “Alert” data is stored, say, every ten minute or every hour. The information in this part may not be delivered depending on the communication setting of the program.

Data sent via the IDMEF format have several attributes as an example shown in Figure 2.6.

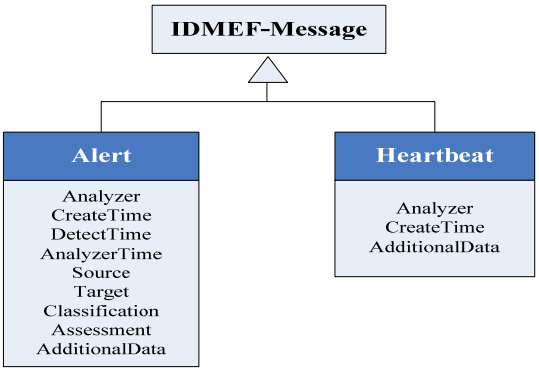

Figure 2.6: Overview of IDMEF Data Model

CHAPTER 3
RELATED WORK

Related research work on this project includes the IDS policy management, alarm correlation and multi sensor agent based IDS. They are described in turn as follows.

3.1 Activeworx IDS policy manager [1, 29]

IDS Policy Manager (IDSPM) of Activeworx.org first released in 2000 was the first rules manager for Snort. Numerous features offered in IDSPM include:

- Ability to disable/enable a rule across all policies
- Ability to use the same policy for multiple sensors
- Ability to add suppress or threshold events from a certain IP or block when viewing a rule
- While updating the policy, rules could be enabled or disabled before they are added to the policy
- Ability to add rules from different update location such as bleedingthreats.com, Snort community and any other locations
- Ability to store all information in a database so that it can work with multiple policies and sensors

IDSPM of Activeworx.org has three steps: (1) Set and update locations for getting Snort rules from Snort.org over HTTP protocol as shown in Figure 3.1; (2) add the policy of a specified version of snort; and (3) add sensors to the policy. In addition, support to download existing policies from sensors and import them can be done via SFTP protocol. Figure 3.2 shows a snapshot of rules management in Snort from IDSPM v.2.

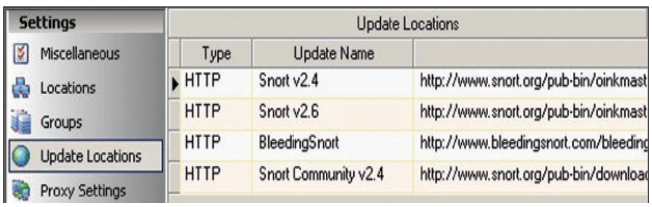

Figure 3.1: Settings and Update Locations of IDSPM v.2

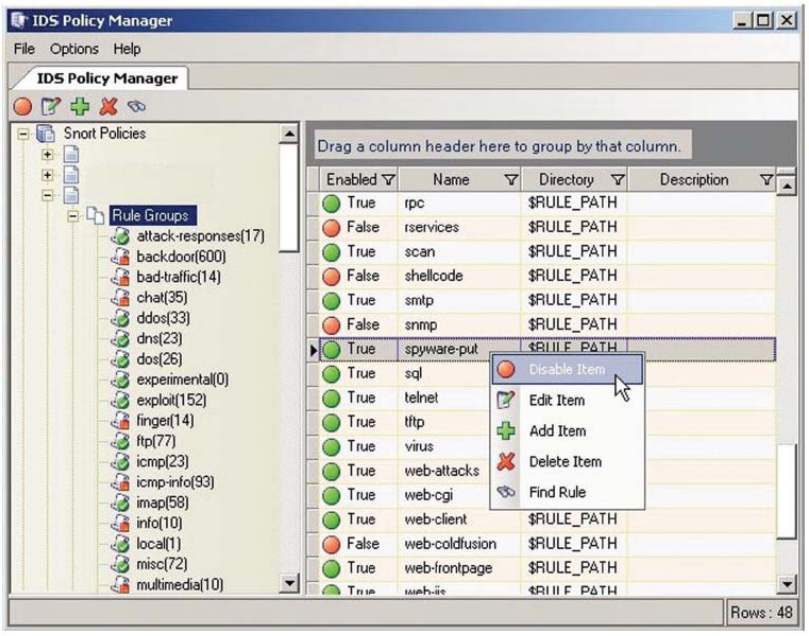

Figure 3.2: Rules Management of IDSPM v.2

3.2 Policy-Controlled Event Management for Distributed Intrusion Detection[21]

The goal of the IDS is to detect malicious activity as quickly as possible in order to allow swift response and provide substantial forensic evidence to understand the damage inflicted. This article presents an extension of Bro's event model that supports scalable policy-controlled distributed event analysis as shown in Figure 3.3. The main idea is to understand the "big picture" activity in a network while each IDS operates individually.

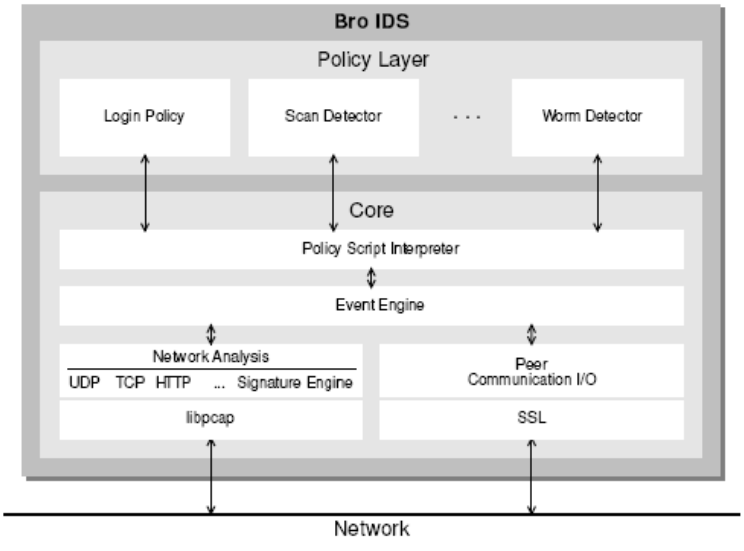

Figure 3.3 Architecture of Event Communication in Bro

The Bro's extension model supports the communication among Bro nodes, and mainly splits event detection mechanisms from event processing policies. An event is generated from the analysis of traffic inputs when corresponding activity is observed. The analysis covers a wide range of network protocols such as RPC, FTP, HTTP, ICMP, SMTP, TCP, UDP, and others. Events are triggered when interesting activities are happened. For example, when a new TCP connection is established or an HTTP request is made. Bro uses the signature detection mechanism and an event is triggered whenever a signature matches. That event is passed to its corresponding policy that will process the event.

Each Bro node has its policy configuration to define its own local policy that may be varied from node to node. The basic building blocks of a policy are event handlers which process events incurred by a node's security policy. Bro allows the transmission of any kinds of state between Bro instances. It realizes *independent* state accumulated at the policy layer and dispersed throughout the network. In other words, state of all Bro nodes could be securely exchanged. Thus, events happened at each node could be communicated, but the identities of each node are controlled tightly.

Basically, an event in Bro has a type consisting of a name and the sequence and types of events' arguments. The values of an event's arguments are distributed and are not bound to a particular node's address space. Events are also available to event handlers while instances of event contain parameters such as time of creation, and which Bro node that triggers the event. In addition, Bro also specify how events are dispatched, and who will see the events. The mechanisms include full broadcast, limited broadcast excluding the originator, an arbitrary subgroup, and the originator only. Hence, Bro does not guarantee that every node sees every instance of event types since it depends on the policy of the dispatching node.

In summary, the extended Bro model provides the flexibility and control necessary to create a large scale distributed IDS with highly configurable distributed event processing.

3.3 Alarm Correlation [18]

Alarm correlation is a mechanism to describe the relationship and co-dependencies between alarms. For clear understandings, alarms and faults are not the

same thing. An alarm is the notification of an event predefined whereas a fault is a disorder in hardware or software within the managed network.

Alarm correlation is a conceptual interpretation of multiple alarms such that new meanings of alarms are discovered. The generic processes for alarm correlation of different network management are given as follows.

- *Compression*: The reduction of repeating and multiple occurrences of an alarm into a single alarm.
- *Count*: The substitution of a specified number of occurrences of alarms with a new alarm.
- *Suppression*: To leave a low-priority alarm in the presence of higher-priority alarm.
- *Boolean*: To eliminate a set of alarms satisfying a Boolean pattern with a new alarm.
- *Generalization*: To refer an alarm via alarms relation.

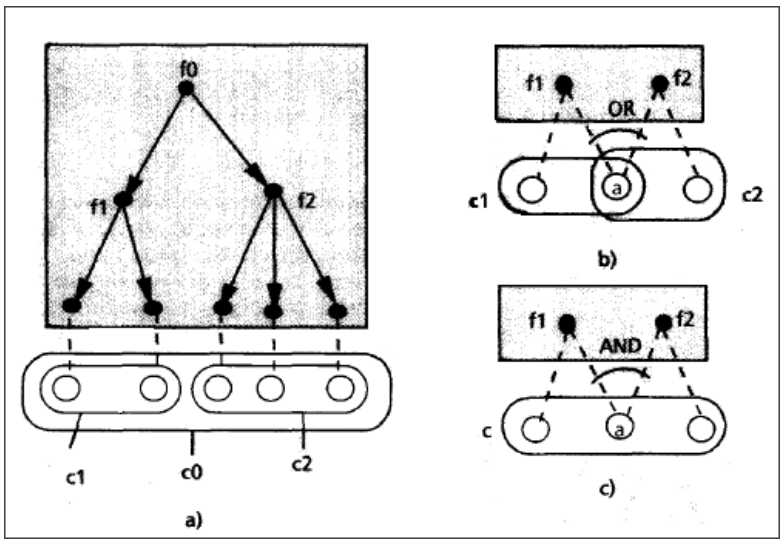

Figure 3.4: Various Alarm Correlations

One major application of alarm correlation is network fault diagnosis. However, not all faults exhibit alarms. Some faults could be recognized indirectly by correlating available alarms. Figure 3.4a illustrates that Correlation c1 detects Fault f1, and Correlation c2 detects Fault f2. Correlating c1 and c2 into Correlation c0 allows the diagnosis of Fault f0. Correlation between alarms due to a common fault is a transitive, reflexive and symmetric relation. If a single alarm is a manifestation of

multiple faults, this relation may not hold. For example, if the alarm shown in Figure 3.4b is caused by Fault f1 or Fault f2, but not both, then Correlation c1 and c2 are formed with a common component alarm, and consequently the correlation relation is not transitive. If the alarm shown in Figure 3.4c is caused by both faults f1 and f2, the diagnosis remains ambiguous as they indicate either a common primary fault or two independent faults of f1 and f2. In order to disambiguate these two cases, additional information is required.

Alarm correlation is based on the principle of model-base reasoning, and its conceptual framework contains two components: structural and behavioral components as shown in Figure 3.5. The structural component describes the network elements, their connectivity and containment relations, and the behavioral component describes alarms and correlations.

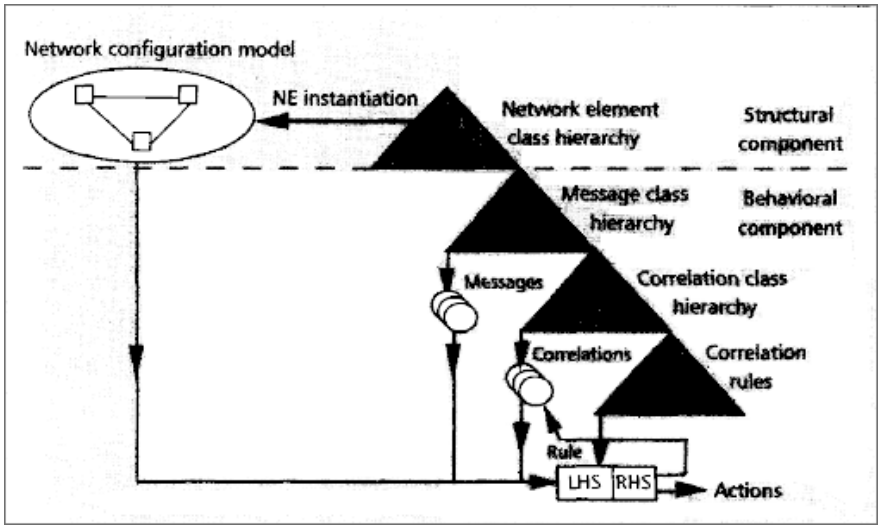

Figure 3.5: Conceptual Framework of Alarm Correlation

The structural component is the description of the managed network. It contains two major parts: the network configuration model and the network element class hierarchy. The network configuration model describes the managed objects or network elements, and the connectivity and containment relations between them. The network element class hierarchy describes the network element type and the class relationships between types. Each network element in the network configuration model is an instance of terminal network element class from the network element class hierarchy.

The behavioral component describes the dynamics of alarm correlation. It contains three major components: the message class hierarchy, the correlation class hierarchy, and correlation rules. The message class hierarchy describes the messages generated by network elements. It is used to control the alarm message-parsing process described more in [19]. The correlation classes are the generalized description of the network state based on the interpretation of network events. The correlation rules recognize events and assert or clear correlation. Different correlation rules may assert or clear the same type of correlation. The conditional part of a rule is a Boolean pattern built upon primary terms and relations.

3.4 Multi-Sensor Agent-Base Intrusion Detection System [27]

This article discusses the implementation of multi-sensor agent-based IDS and the framework to support the analysis and response as shown in Figure 3.6. They use data in tcpdump format for interpreting network traffic. Specifically, they are interested in analyzing abnormal or malicious traffic.

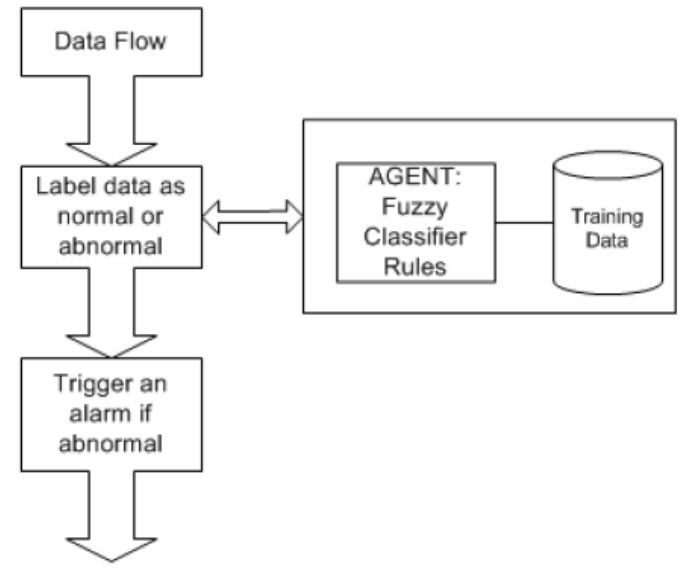

Figure 3.6: Multi-Sensor Agent-Based IDS Architecture

The system is implemented based on Snort and focused on easy navigation and reporting. They use the following open software for building the prototype.

- *Tcpdump*: Collecting packet contents for analysis. Packets can be saved and replayed via traffic analysis tools such as Ethereal.

- *Snort with mySQL*: Keeping detection intrusion events to the database for analyzing them later.
- *Trafd*: Collecting data to memory and dump results periodically.
- *Trafshow*: Displaying real time statistics on collected data, and checking what flow consuming bandwidth in reactive mode.

The heart of this model is to control the inference engine using a set of rules consisting of linguistic variables and associated fuzzy functions. The differences between the normal and abnormal activities are not distinct, but rather fuzzy since the fuzzy inference module can help reduce false alarms in determining intrusive actives.

CHAPTER 4
PROPOSED WORK

This chapter presents the proposed model and its architecture. We also describe database tables used in this work, and explain the system processes and modules in detail.

4.1 Overview of the Proposed Model

This work proposes a distributed IDS model which suites the needs of Mahidol University network. The model mainly consists of two components: IDS sensor and IDS manager as shown in Figure 4.1. IDS sensors are Snort machines installed at various sites across Mahidol campuses, and they are mostly at the computer labs since students and faculty staffs often access the Internet from there. IDS manager is a centralized node responsible for collecting log analysis information from IDS sensors. It also analyzes the use of Snort rules at all IDS sensors, determines the number of false alarms, and finally updates the latest signatures to all IDS sensors.

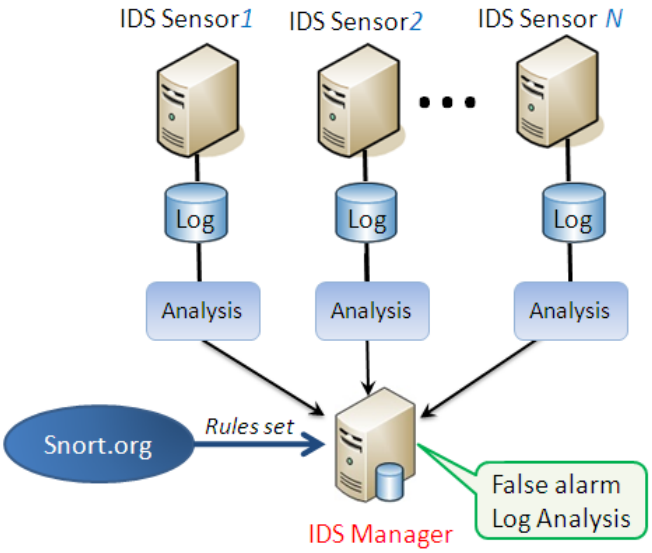

Figure 4.1: Overview of the Proposed Model

If there is an attack at a local network, the local IDS sensor will detect and generate the corresponding alert logs. All alert logs are then kept in the local database.

In addition, all Snort rules will be monitored for their deployment. In other words, an alert log generated by a Snort rule indicates that such rule is used. Similarly, if there is no alert log generated by some rules, then such rules are never used and should not be uploaded to that Snort machine. If the number of active rules is kept minimized, the Snort performance would be improved. Initially, all IDS sensors will be installed with the full set of Snort rules. After monitoring network traffic for a period of time and some logs are generated, the system will analyze those logs and summarize which rules are active as well as report both alert logs and rule usage to the IDS manager.

Hence, we need two additional components: the databases at both IDS sensors and the IDS manager, and the mechanism for communicating between IDS sensors and the IDS Manager. The details of IDS sensor and IDS Manager are explained in the next section.

4.2 IDS Sensors

IDS sensor implemented as Snort is installed at each local network, and responsible for signature detection for network attacks which may come from users or machines in a lab infected with viruses that tried to send attacking traffic to the outside world. In addition to Snort's typical functions of intrusion detection and collecting logs, the IDS sensor must perform log analysis in order to monitor the type of attacks detected and to identify which rules are locally active. Later, the IDS sensor sends such information to the IDS manager, and also receives the latest rules updates from the IDS manager downloading from *snort.org*. Thus, an IDS sensor performs three processes: Intrusion Detection, Log Analysis, and Rules Update. All processes are shown in the diagram of Figure 4.2, and the details of each process are described as follows.

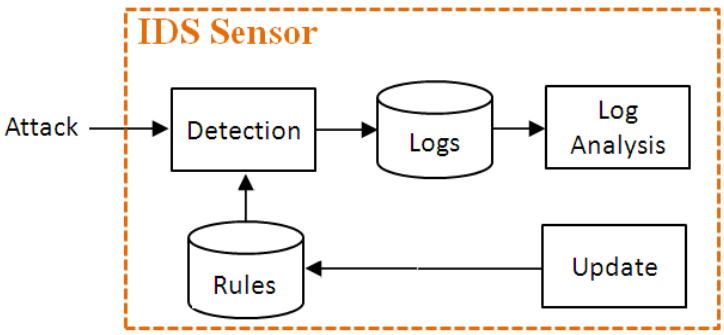

Figure 4.2: IDS Sensor Diagram

(1) Intrusion Detection Process

Intrusion detection is a typical IDS process. Incoming packets are captured and examined whether their payload are matched with some predefined rules. If they are matched, the IDS will record such attack events and generate alert logs. Those logs will be analyzed in the next process.

(2) Log Analysis Process

Only five interested attributes are selected from the whole record of Snort logs. They are listed below.

- *Rule ID:* Signature or Rule ID that identifies the type of intrusion
- *Source IP*: IP address of sender
- *Destination IP*: IP address of receiver
- *Occurrences*: The number of occurrences of detected intrusion type
- *Attack Date and Time*: Date and time of intrusions

Attack date and time tells us the time of the attack. Rule ID represents the rule matched with the packet. Source IP is the IP address of the intrusion packet and Destination IP is the IP address for the victim host. At the end of the day, alert logs with such five attributes and rules usage will be sent to the IDS manager. Figure 4.3 shows a sample of alert logs and Figure 4.4 shows a sample output of log analysis for rules usage.

Rule_ID	IP_SRC	IP_DST	Occurrences	Time
466	10.22.5.4	10.22.5.162	5	2007-09-17 09:17:40
2049	10.22.5.27	255.255.255.255	3	2007-09-17 08:23:25
466	10.22.5.54	10.22.5.57	4	2007-09-16 18:36:13
2049	10.22.5.57	255.255.255.255	1	2007-09-17 09:06:42
2049	10.22.5.59	255.255.255.255	1	2007-09-17 09:40:09
466	10.22.5.65	10.22.5.27	1	2007-09-17 08:59:05
2049	10.22.5.69	255.255.255.255	2	2007-09-17 07:55:38
2049	10.22.5.71	255.255.255.255	3	2007-09-17 08:27:09
466	10.22.5.105	10.22.5.65	6	2007-09-17 08:18:01
466	10.22.5.112	10.22.5.123	4	2007-09-17 06:49:01

Figure 4.3: Sample Alert Logs of Selected Attributes

(3) Rules Update Process

This process will listen to the update signal from the IDS Manager. When the IDS sensor receives new Snort rules, it will reread the Snort configuration and process all rules again. Thus, Snort rules at IDS sensors are kept up to date.

IP_SRC	Rule_Name	Occurrences
10.22.51.194	TFTP Get	1008
10.22.51.254	ICMP Destination Unreachable Communication Adminis...	700
202.28.166.38	SNMP trap udp	138
202.28.162.1	ICMP Destination Unreachable Communication with De...	97
10.22.51.254	ICMP Destination Unreachable Communication Adminis...	92
10.27.1.243	SNMP trap udp	78
10.22.5.254	ICMP Destination Unreachable Communication Adminis...	54
10.22.51.77	MS-SQL ping attempt	36
10.22.51.206	MS-SQL ping attempt	36

Figure 4.4: Sample Outputs of Log Analysis at IDS Sensor

4.3 IDS Manager

The IDS manager consists of four processes: log parser, log analysis, rules management and web interface. The diagram of the IDS manager is shown in Figure 4.5, and each process is described as follows.

(1) Log Parser Process

When the IDS manager receives a huge number of logs from all IDS sensors, it needs to transform such logs to the correct format before putting them into the database for further analysis. The format of a log from an IDS sensor is as follows.

Format: [signature] [source IP] [destination IP] [yyyymmddHHMMss]

Example: 466 3390875174 169215480 20070925140933

Explanation: This log is recorded on September 25, 2007 at time 14:09:33 from the source IP “3390875174” or 202.28.166.38 to the destination IP “169215480” or 10.22.5.248 with the signature id of 466.

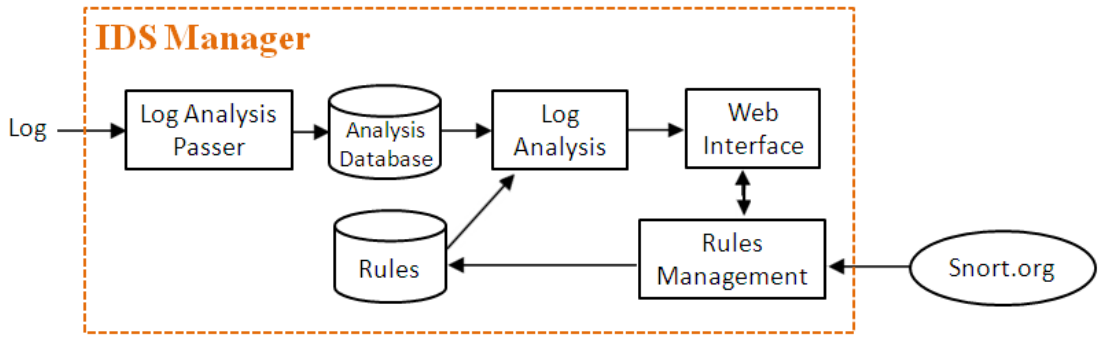

Figure 4.5: Diagram of IDS Manager

(2) Log Analysis Process

The log analysis process is responsible for analyzing intrusion patterns based on rules usage information. Figure 4.6 shows the accumulated number of occurrences in ascending order for all rules received from IDS sensors for a few days.

Rule_Name	Occurrence	Sensor_ID
SNMP public access udp	72380	4
SNMP request udp	72380	4
SNMP trap udp	7681	4
ICMP Destination Unreachable Communication	5674	2
MISC UPnP malformed advertisement	5283	1
ICMP Destination Unreachable Communication	2878	4
ICMP L3retriever Ping	2121	4
TFTP Get	1847	2
ICMP Destination Unreachable Communication	952	4
ICMP Destination Unreachable Communication	783	1
SCAN UPnP service discover attempt	759	4
ICMP Destination Unreachable Communication	651	3
TFTP Get	648	3
ICMP Destination Unreachable Communication	535	2
MS-SQL ping attempt	483	4
MS-SQL ping attempt	471	3

Figure 4.6: Sample Outputs of Log Analysis at IDS Manager

(3) Rule Management Process

The rule management process is responsible for updating rules for all IDS sensors and make sure that all rules are consistent. However, at any moment, each IDS sensor may have different active rules. The IDS manager could decide which rules should be active at which IDS sensor. It is also responsible for downloading the latest update of rules from the Snort site (www.snort.org), and for sending such updates to IDS sensors either manually or automatically. For an automatic method, IDS sensors run a *cron* job at a specific time to download the latest update rules from the IDS manager, and restart themselves to make such rules updated and active.

(4) Web Interface Process

This web interface process is responsible for handling all kinds of queries regarding logs collected from IDS sensors and displaying significant information from log analysis. The interface shown in Figure 4.7 allows the system administrator to understand log information and be able to visualize intrusion activities in our network.

Hence, the system administrator could decide which rules would be active and which rules would be removed at IDS sensors. In addition, the system would help update new rules to IDS sensors automatically.

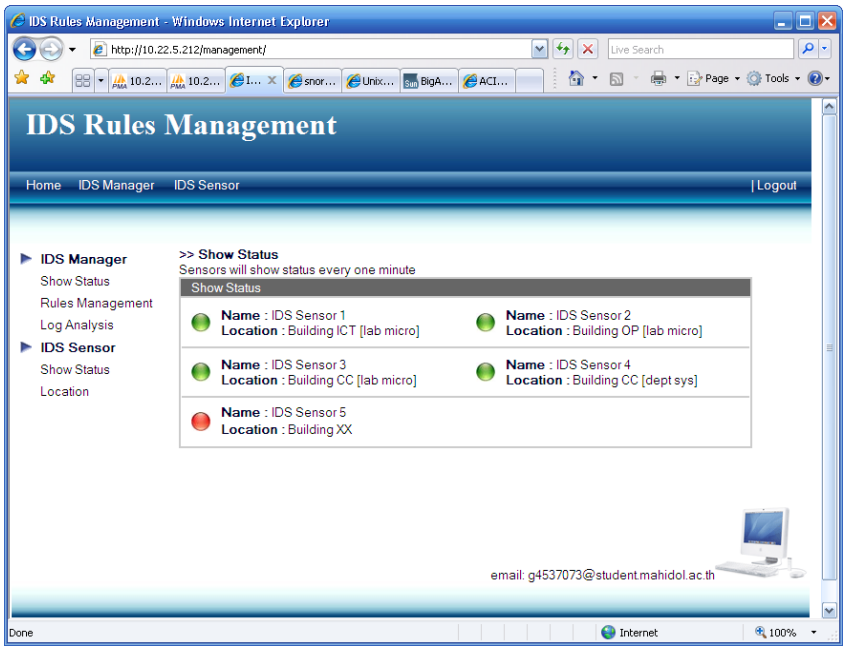

Figure 4.7: Web Interface for IDS Manager

4.4 Database Design

In this work, two databases are used. One database is at each IDS sensor and the other is at the IDS manager. These two databases are related to intrusion logs collected by Snort IDS. We discuss them in detail as follows.

4.4.1 IDS Sensor Log Database

The log database at IDS sensors keeps all intrusion log records. The snort database schema shown in Figure 4.8 is used for collecting logs from IDS sensors. There are 19 tables in the schema, and the description of all tables is summarized in Table 4.1. However, only some tables are interested to our analysis.

Every packet generating an alert log is kept in the Event table while the packet details are kept in the Data and Opt tables. The IP address of the packets and the packet header information are placed in the IPhdr table which refers to other protocol tables such as ICMPhdr, TCPhdr and UDPhdr tables. Each event or packet detected refers to rule or signature ID differently for each IDS sensor.

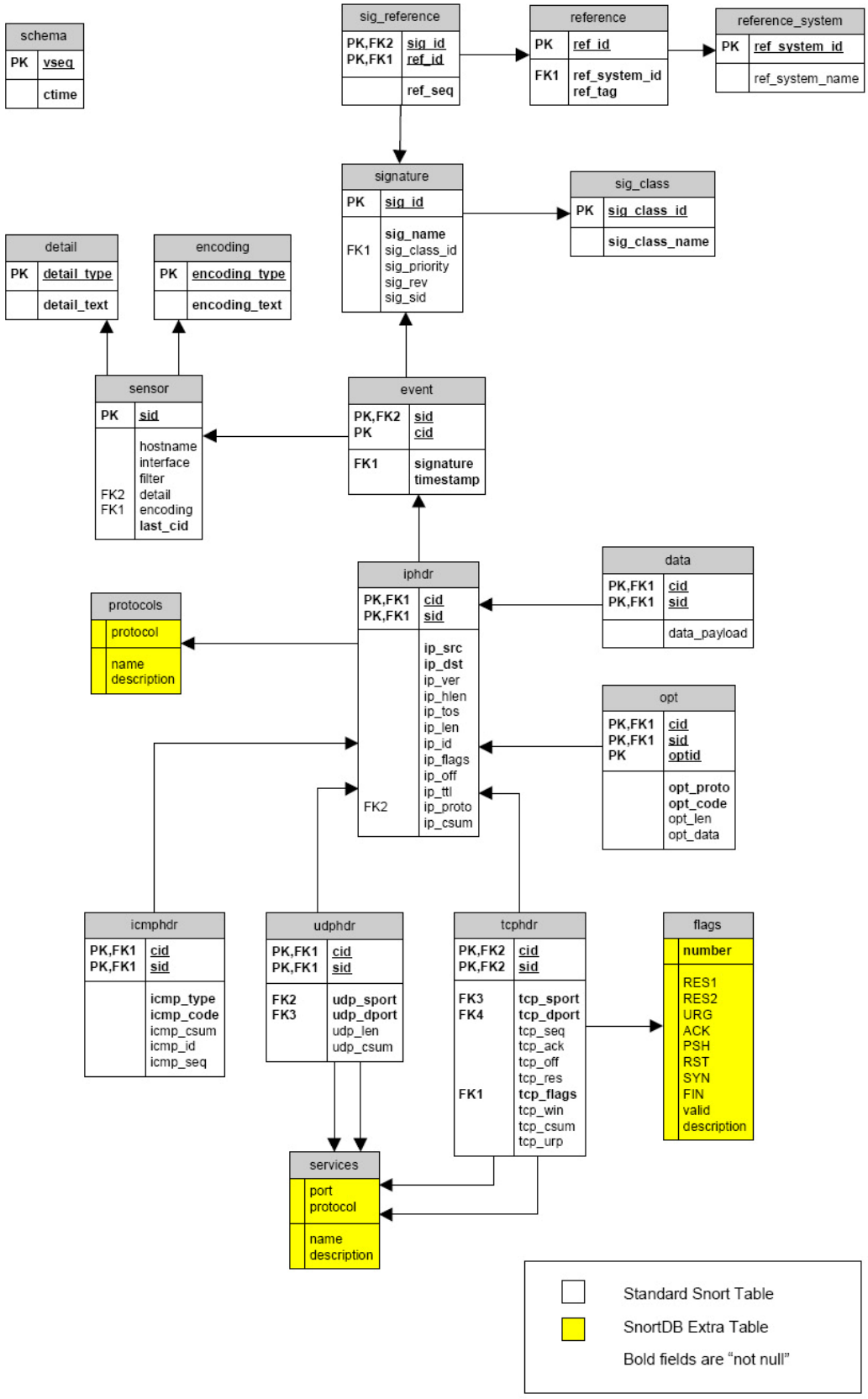

Figure 4.8: Snort Log Database Schemas

Table Name	Table Description
Data	Details of intrusion packets
Detail	Types of record logs: Fast and Full
Encoding	Types for encoding packets: Hex, Base64 and ASCII
Event	Data of intrusion packets such as detecting rules, detected time
ICMPhdr	Information of ICMP packet separated from IPhdr
IPhdr	Information of packets such as source IP and destination IP
Opt	Additional packet information separated from IPhdr
Reference	Reference of each signature (Rule)
Reference_system	Reference system list (lookup table for Reference table)
Schema	Self-documented information about the database
Sensor	Details of IDS sensor such as hostname and interface used
Sig_class	Normalized listing of signature classifications
Sig_reference	Reference information for a signature
Signature	Signature names of use, order from found
TCPhdr	Information of TCP packet separated from IPhdr
UDPhdr	Information of UDP packet separated from IPhdr
Flags	Additional information of TCPhdr
Protocols	Additional information of IPhdr
Services	Additional information of UDPhdr and TCPhdr

Table 4.1: Snort Log Database Schemas

Table: Data
Description: Data payload of an intrusion packet

Field	Type	Null	Key	Default	Description
sid	int(10)	NO	PRI		Sensor ID
cid	int(10)	NO	PRI		Event ID
data_payload	text	YES		NULL	Packet payload encoded according to sensor.encoding

Table: Detail

Description: Type of log database has the following two options. The default is full.

- *Fast*– Record only the minimum amount of data. The following logged fields are are timestamp, signature, source IP, destination IP, source port, destination port, TCP flags, and protocol.
- *Full* – Record all details of a packet that caused an alert including TCP/IP options and the payload.

Field	Type	Null	Key	Default	Description
detail_type	int(3)	NO	PRI		Detail type
detail_text	text	NO			Detail name

Table: Encoding

Description: Encoding option of data has the following three options. The default is Hex.

- *Hex* – Representing binary data as a hex string.
- *Base64* – Representing binary data as a base64 string.
- *Ascii* – Representing binary data as an ascii string. This is the only option where will lose data. (Snort suggestion).

Field	Type	Null	Key	Default	Description
encoding_type	int(3)	NO	PRI	0	Encoding type
encoding_text	text	NO			Encoding name

Table: Event

Description: Metadata about the alert that stores event when intruder packet is found, and record the referenced rule in the signature field.

Field	Type	Null	Key	Default	Description
sid	int(10)	NO	PRI	0	Sensor ID
cid	int(10)	NO	PRI	0	Event ID
signature	int(10)	NO	FK	0	Signature ID
timestamp	datetime	NO		0000-00-00 00:00:00	Timestamp when the event was logged

Table: ICMPhdr

Description: ICMP protocol fields.

Field	Type	Null	Key	Default	Description
sid	int(10)	NO	PRI	0	Sensor ID
cid	int(10)	NO	PRI	0	Event ID
icmp_type	int(3)	NO		0	ICMP type
icmp_code	int(3)	NO		0	ICMP code
icmp_csum	int(5)	YES		Null	ICMP checksum
icmp_id	int(5)	YES		Null	ICMP ID
icmp_seq	int(5)	YES		Null	ICMP sequence number

Table: IPhdr

Description: IP protocol fields.

Field	Type	Null	Key	Default	Description
sid	int(10)	NO	PRI	0	Sensor ID
cid	int(10)	NO	PRI	0	Event ID
ip_src	int(10)	NO		0	Source IP address (32-bit unsigned int)
ip_dst	int(10)	NO		0	Destination IP address (32-bit unsigned int)
ip_ver	int(3)	YES		Null	IP version
ip_hlen	int(3)	YES		Null	IP Header length
ip_tos	int(3)	YES		Null	IP type-of-service
ip_len	int(5)	YES		Null	IP datagram length
ip_id	int(5)	YES		Null	IP ID
ip_flags	int(3)	YES		Null	IP flags
ip_off	int(3)	YES		Null	IP fragment offset
ip_ttl	int(3)	YES		Null	IP time-to-live
ip_proto	int(3)	YES		Null	IP protocol
ip_csum	int(5)	YES		Null	IP checksum

Table: Opt

Description: IP and TCP options.

Field	Type	Null	Key	Default	Description
sid	int(10)	NO	PRI	0	Sensor ID
cid	int(10)	NO	PRI	0	Event ID
optid	int(10)	NO	PRI	0	Option ID (multiple options per alert)
opt_proto	int(3)	NO		0	Option protocol (IP, TCP)
opt_code	int(3)	NO		0	Option code
opt_len	int(5)	YES		Null	Option length
opt_data	text	YES		Null	Option data

Table: Reference

Description: Snort Reference IDS for a signature.

Field	Type	Null	Key	Default	Description
ref_id	int(10)	NO	PRI	0	Reference ID
ref_system_id	int(10)	NO			Reference system ID
ref_tag	char(20)	NO			Reference tag

Table: Reference_system

Description: Reference IDS for a table reference which describes reference name.

Field	Type	Null	Key	Default	Description
ref_system_id	int(10)	NO	PRI	0	Reference system ID
ref_system_name	char(20)	YES		Null	Reference system name

Table: Schema

Description: Self-documented information about the database.

Field	Type	Null	Key	Default	Description
vseq	int(10)	NO	PRI	0	Database schema ID number (e.g. '102')
ctime	datetime	NO		0000-00-00 00:00:00	Timestamp of database creation time

Table: Sensor

Description: Sensor name.

Field	Type	Null	Key	Default	Description
sid	int(10)	NO	PRI	0	Sensor ID
hostname	text	YES		Null	Hostname of the sensor
interface	text	YES		Null	Network interface
filter	text	YES		Null	BPF filter
detail	int(4)	YES		Null	Detail level of the logging
encoding	int(4)	YES		Null	Encoding format of the payload

Table: Sig_class

Description: Normalized listing of alert/signature classifications.

Field	Type	Null	Key	Default	Description
sig_class_id	int(10)	NO	PRI	0	Signature classification ID
sig_class_name	char(60)	NO	PRI	0	Classification name

Table: Sig_reference

Description: Reference information for a signature.

Field	Type	Null	Key	Default	Description
sig_id	int(10)	NO	PRI	0	Signature ID
ref_seq	int(10)	NO	PRI	0	Reference sequence number (multiple references)
ref_id	int(10)	NO			Reference ID

Table: Signature

Description: Normalized listing of alert/signature names, priorities, and IDS revision.

Field	Type	Null	Key	Default	Description
sig_id	int(10)	NO	PRI	0	Signature ID
sig_name	char(255)	NO			Signature Name
sig_class_id	int(10)	YES		NULL	Classification ID
sig_priority	int(10)	YES		NULL	Priority
sig_rev	int(10)	YES		NULL	Revision number
sig_sid	int(10)	YES		NULL	Internal signature ID

Table: TCPhdr

Description: TCP protocol fields.

Field	Type	Null	Key	Default	Description
sid	int(10)	NO	PRI	0	Sensor ID
cid	int(10)	NO	PRI	0	Event ID
tcp_sport	int(5)	NO		0	TCP source port
tcp_dport	int(5)	NO		0	TCP destination port
tcp_seq	int(10)	YES		Null	TCP sequence number
tcp_ack	int(10)	YES		Null	TCP ACK number
tcp_off	int(3)	YES		Null	TCP offset
tcp_res	int(3)	YES		Null	TCP reserved
tcp_flags	int(3)	YES		Null	TCP flags
tcp_win	int(5)	YES		Null	TCP window
tcp_csum	int(5)	YES		Null	TCP checksum
tcp_urp	int(5)	YES		Null	TCP urgent pointer

Table: UDPhdr

Description: UDP protocol fields.

Field	Type	Null	Key	Default	Description
sid	int(10)	NO	PRI	0	Sensor ID
cid	int(10)	NO	PRI	0	Event ID
udp _sport	int(5)	NO		0	UDP source port
udp _dport	int(5)	NO		0	UDP destination port
udp_len	int(5)	YES		Null	UDP length
udp_csum	int(5)	YES		Null	TCP ACK number

Table: Flags

Description: Referenced from TCPhdr table, and provides an association between TCP flags in decimal with specific flags.

Field	Type	Null	Key	Default	Description
number	int(10)	NO	PRI	0	TCP flags number
RES1	int(10)	YES			1 if RES1 bit is active
RES2	int(10)	YES			1 if RES2 bit is active
URG	int(10)	YES			
ACK	int(10)	YES			
PSH	int(10)	YES			
RST	int(10)	YES			
SYN	int(10)	YES			
FIN	int(10)	YES			
Valid	int(10)	YES			1 if combination is valid
description	char(255)	YES		NULL	Common description if available

Table: Protocols

Description: An extra table referenced form IPhdr table and provides an association between protocol number and name

Field	Type	Null	Key	Default	Description
protocol	int(3)	NO	PRI		The protocol number
name	char(50)	NO			The protocol name
description	char(50)	NO			Protocol description

Table: Services

Description: Referenced from UDPhdr or TCPhdr for keeping TCP and UDP service list. It provides an association between port number, protocol, and service.

Field	Type	Null	Key	Default	Description
port	int(3)	NO	PRI		The port number
protocol	int(3)	NO	PRI		The protocol number
name	char(50)	NO			The common service on that port
description	char(255)	NO			A description of the service

4.4.2 IDS Manager Snort Log Database

Figure 4.9 shows the schema of the snort analysis database used for log analysis log at the IDS manager. The descriptions of each table in the database are given in Table 4.2.

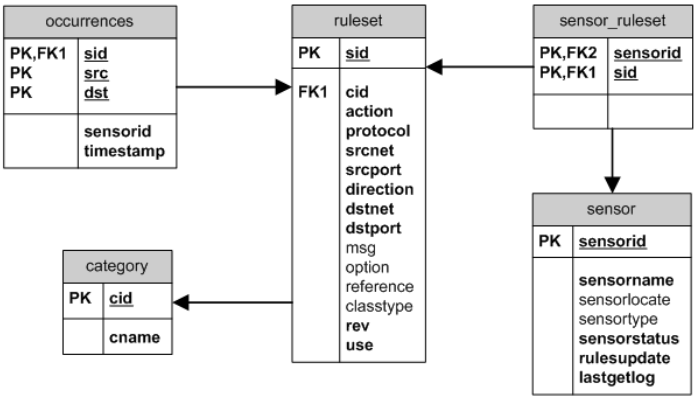

Figure 4.9: Snort Analysis Database Schemas

Table Name	Table Description
Category	Category of Snort rules
Occurrences	Logs from IDS sensors
Ruleset	Rules set of IDS sensor
Sensor	Sensor information
Sensor_ruleset	Rule set for each sensor

Table 4.2: Snort Log Analysis Database Description

Table: Category

Description: Rule Category.

Field	Type	Null	Key	Default	Description
cid	int(10)	NO	PRI		Category ID
cname	text	NO			Category name

Table: Occurrences

Description: Number of occurrences of intrusions found with source, destination and rule.

Field	Type	Null	Key	Default	Description
sid	int(10)	NO	PRI		Rule ID
src	int(10)	NO	PRI		Source IP address (32-bit unsigned int)
dst	int(10)	NO	PRI		Destination IP address (32-bit unsigned int)
sensor_id	int(10)	NO			
timestamp	datetime	NO			

Table: Ruleset

Description: Set of Rules of IDS maintained by the IDS Manager

Field	Type	Null	Key	Default	Description
sid	int(10)	NO	PRI	0	Rule ID
cid	int(10)	NO		0	Category ID
action	char(20)	NO		Alert	Action of rule
protocol	char(20)	NO		TCP	Protocol of rule
srcnet	char(40)	NO		Any	Source IP or network
srcport	char(20)	NO		Any	Source port
direction	char(2)	NO		->	Direction of Rule
dstnet	char(40)	NO		Any	Destination IP or network
dstport	char(20)	NO		Any	Destination port
msg	text	YES		Null	Rule name
option	text	YES		Null	Rule option
reference	text	YES		Null	Reference of rule
classtype	char(40)	NO		0	Classtype of rule
rev	int(4)	NO		1	Revision of rule
use	boolean	NO		1	Boolean show this rule is use or not.

Table: Sensor

Description: Store information of IDS sensor and their details.

Field	Type	Null	Key	Default	Description
sensor_id	int(10)	NO	PRI		Sensor id
sensor_name	text	YES			Sensor name
sensor_locate	text	YES			Sensor location
sensor_type	text	YES			Sensor type (e.g. Snort)
sensor_status	boolean	NO		1	Sensor status
ruleupdate	datetime	NO			Last of update rules
lastgetlog	datetime	NO			Last to get log

Table: Sensor_ruleset

Description: Information about rules set for each IDS sensor.

Field	Type	Null	Key	Default	Description
sensor_id	int(10)	NO	PRI		Sensor id
sid	int(10)	NO	PRI		Rule ID

4.5 Data Analysis

In this part, we focus on the analysis at the IDS manager. They are the analysis of rule usage and the percentage of false alarms.

(1) Rule Usage Analysis

All collected logs from IDS sensors will be analyzed and presented in a simple view based on the values contained in the log database. The results include which rules are the most popular attack, which IDS sensor has the highest intrusion attempts, and other useful information.

(2) False Alarm Analysis

False alarms or false positives are alerts that Snort misclassifies them as intrusion attempts. Thus, they are benign traffic and can safely be ignored. IDS system should learn how to recognize such false alarms and disable them accordingly so that the accuracy of detection would be higher.

A simple way to detect false alarms is to look at alert statistics. For example, if a large number of specific alerts are generated higher than other alerts, then it is likely that some traffic is misclassified. Thus, the system should examine the source and destination IP addresses. If both addresses are the servers within our network, such traffic may simply be normal traffic coming from a server via a unique port or protocol just for monitoring the network. If the traffic is likely to be intrusion attempts, then we need to correct the problem at the source host.

CHAPTER 5
IMPLEMENTATION AND EXPERIMENTS

This chapter describes the implementation of our proposed model. We explain hardware and software tools used in the experiments, and describe the details of the experiments.

5.1 System Implementation

The system consists of two components communicating with each other. They are IDS sensors and the IDS manager. Figure 5.1 gives an overview of our system implementation. Four IDS sensors are installed across campuses of Mahidol University network.

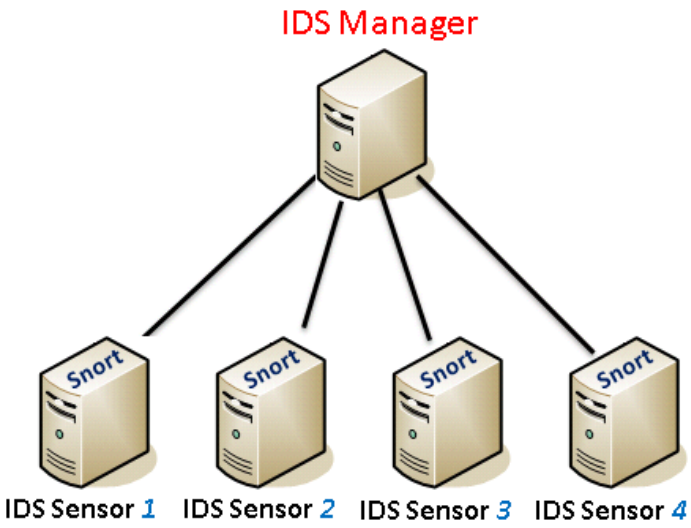

Figure 5.1: System Implementation of the Proposed Model

5.2 Hardware and Software Configuration

Table 5.1 shows the hardware and software configuration used in our implementation and experiments. The hardware configuration of all machines is similar; thus, it should not affect the overall performance. But, the software configuration of IDS sensors and the IDS manager are different. We describe each software component as follows.

- Apache: Web server for the IDS manager. It supports Perl and php languages used for creating the web interface of log analysis and rules management.
- Libxml: The library for parsing XML documents used by Snort to create alert outputs in XML form.
- Libidmef: Libidmef is built on Gnome's libxml. It provides C structure implementations of XML documents and data. Among many attractive attributes, libxml allows for XML validation with DTD's. In other words, libidmef provides a simple front end to libxml.
- Libpcap: The library for packet capture used by Snort to capture packets within a network.
- MySQL: Database server for IDS sensors and the IDS manager.
- PCRE: PCRE stands for Perl Compatible Regular Expressions. It is an open source library written in C and Snort uses it for matching rules with transmitting packets on a network.
- PHP: A programming language that allows us to create dynamic contents interacting with the databases. PHP is used for developing web based software applications for IDS manager.
- PhpMyAdmin: An open source program in PHP script used to manage MySQL databases. We also use it to manage and analyze alert logs.
- Snort: Open source IDS used as IDS sensor has features such as packet sniffer to monitor network traffic in real time, and scrutinize each packet closely to detect a dangerous payload or suspicious anomalies.

In addition, each IDS sensor has been configured to support mysql and idmef features.

5.3 Implementation of IDS Sensors

IDS sensor use Snort version 2.4 with Mysql database V5.0. Other library supports are libpcap, pcre, libxml, and libidmef. Moreover, shell scripts are written to implement some processes in IDS sensor. We describe each process as follows.

	IDS Sensors				IDS Manager
	ICT LAB	**OP LAB**	**CC LAB**	**System LAB**	
CPU	Intel P.4 2.7GHz	Intel P.3 1GHz	Intel P.3 1GHz	Intel P.3 1GHz	Intel P.4 2.7GHz
RAM	256 MB	256 MB	256 MB	256 MB	256 MB
Hard Disk	40 GB	20 GB	20 GB	20 GB	40 GB
Network	Ethernet 100 Mbps	Ethernet 100 Mbps	Ethernet 100 Mbps	Ethernet 100 Mbps	Ethernet 100 Mbps
OS	Red Hat ES 5	Red Hat ES 5	Fedora Core 5	Red Hat ES 5	Red Hat ES 5
Software Tools	Snort 2.4 Mysql 5.0 libpcap 0.8 pcre 7.2 libxml 2.6 libidmef 1.0	Snort 2.4 Mysql 5.0 libpcap 0.8 pcre 7.2 libxml 2.6 libidmef 1.0	Snort 2.4 Mysql 5.0 libpcap 0.8 pcre 7.2 libxml 2.6 libidmef 1.0	Snort 2.4 Mysql 5.0 libpcap 0.8 pcre 7.2 libxml 2.6 libidmef 1.0	Apache 2.2 Mysql 5.0 PHP 5.1 phpMyAdmin 2.11
IP Address	10.34.11.xx	10.41.51.xx	10.22.51.xx	10.22.5.xx	10.22.5.xx

Table 5.1 Hardware and Software Configuration

5.3.1 Log Collection

Collecting logs at an IDS sensor uses Snort configuration for specifying where to put the output of alert logs. In this work, we specify in the file “snort.conf” to store such logs into the MySQL database having the schema given in Section 4.4.1. The number of alert logs depends on the number of intrusion attempts found within a network, and what matching rules that Snort found. Figure 5.2 shows the Snort configuration to put alert logs to the database.

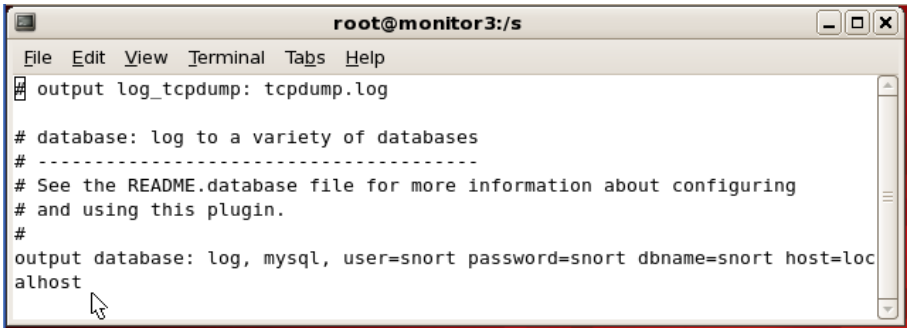

Figure 5.2: IDS Configuration of Snort for Database Output

5.3.2 Preprocessing

Since all alert logs have already been stored in the MySQL database, we need to prepare them into a proper format that is ready to be sent. Only a few interesting attributes such as signature ID, source IP, destination IP, occurrences of intrusions found, and the date and time of intrusions are selected, and gathered into a file before being sent to the IDS manager. Figure 5.3 shows examples of intrusion packets found at one of IDS sensors, and Figure 5.4 shows the executing SQL command for collecting logs from the database.

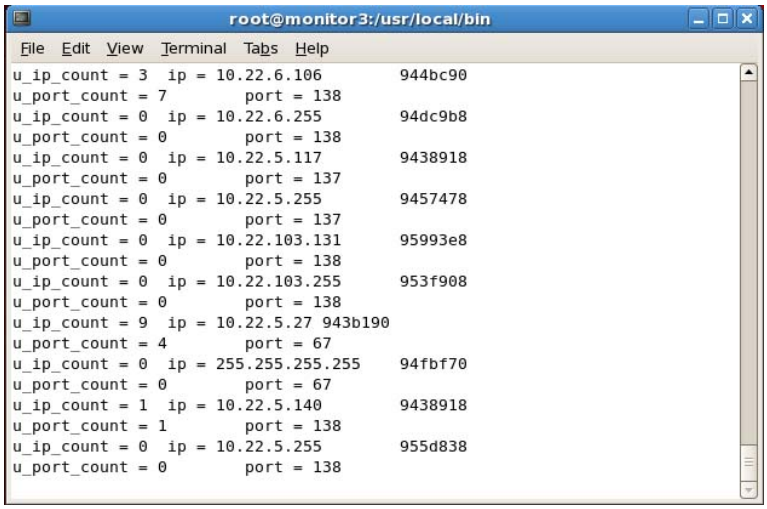

Figure 5.3: Sample Intrusion Packets Found at IDS Sensor

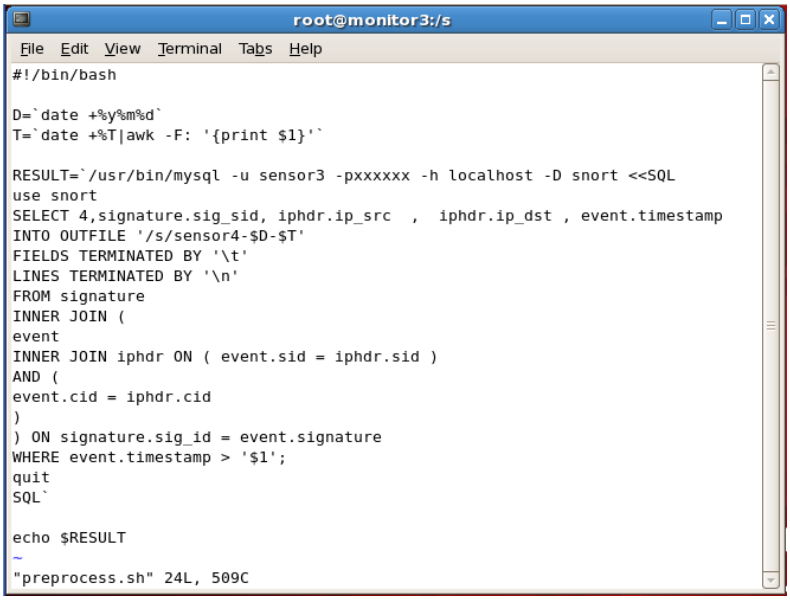

Figure 5.4: SQL Query for Collecting Logs from Database

5.3.3 Sending Logs

A script for a *cron* job shown in Figure 5.5 at IDS sensors would send alert logs to the IDS manager every hour. Such alert logs are sent via the FTP protocol. A shell script is written to control the sending and checking of file transfer. Figure 5.6 shows the shell script for sending alert logs.

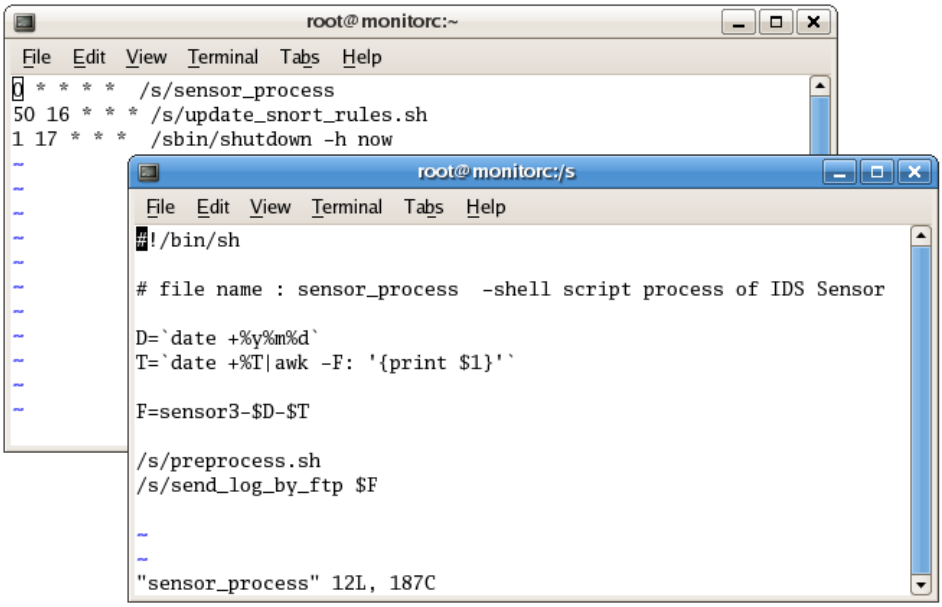

Figure 5.5: Script for the “cron” Job

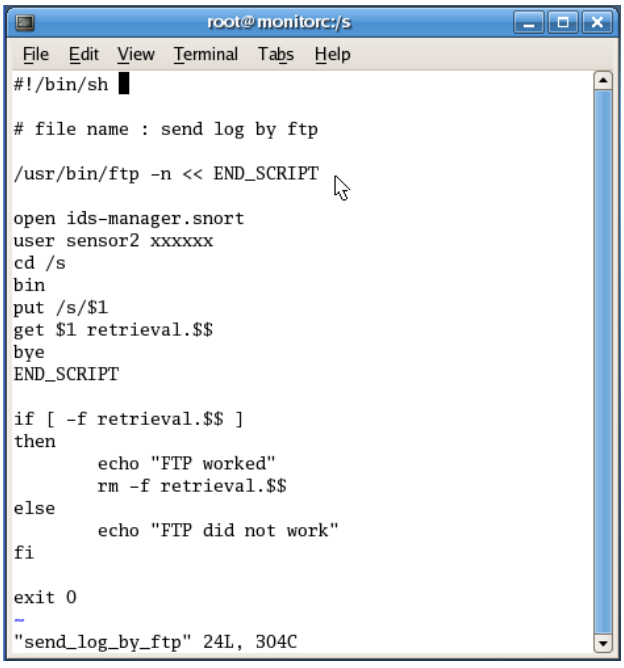

Figure 5.6: Script for Sending Alert Logs

5.3.4 Results of IDS Sensors

The IDS manager can examine the data sent by IDS sensors via querying at the alert logs. Figure 5.7 shows the SQL query for counting intrusion types at every hour of September 19, 2007. Figure 5.8 shows the corresponding output of SQL query in Figure 5.7 as the occurrences of intrusion types at every hour, and Figure 5.9 shows the corresponding output as a graph.

```
SQL query:
SELECT signature, COUNT( sid ) AS occur, HOUR( TIMESTAMP ) AS hourf
FROM event
WHERE TIMESTAMP >= "070919"
AND TIMESTAMP <= "070920"
GROUP BY hourf, signature
LIMIT 0 , 30
```

Figure 5.7: SQL Query for counting Intrusion Types at Every Hour

signature	occur	hourf
1	1	0
2	8	0
3	397	0
4	397	0
5	4	0
6	1	0
1	1	1
2	9	1
3	343	1
4	343	1
5	5	1
6	3	1
2	3	2
3	86	2
4	86	2
2	3	3
2	3	4
⋮		
11	1	21
1	1	22
2	26	22
3	1008	22
4	1008	22
5	10	22
6	4	22
9	2	22
2	12	23
3	663	23
4	663	23
5	2	23

Figure 5.8: The Output Showing Occurrences of Intrusion Types at Every Hour

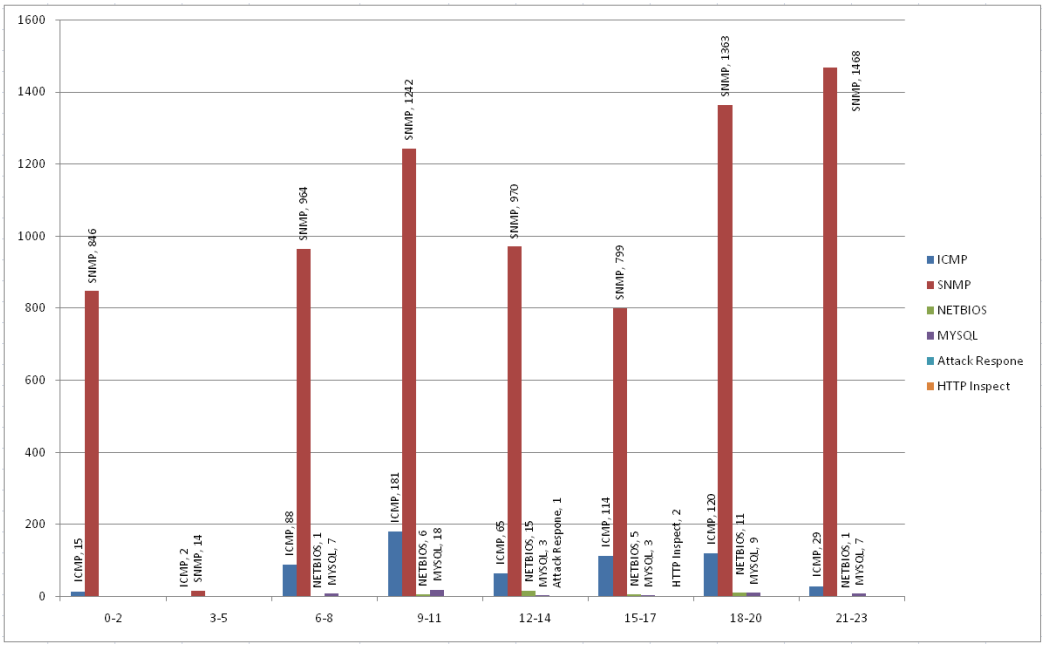

Figure 5.9: The Graph Showing Occurrences of Intrusion Types at Every Hour

5.4 Implementation of IDS Manager

The IDS manager consists of three components: alert log insertion, alert log analysis, reporting, rules updates and false alarms analysis. We describe them in turn as follows.

5.4.1 Alert Log Insertion

The IDS Manager transforms alert logs received from IDS sensors into its own format at a specified time using the script shown in Figure 5.10 below.

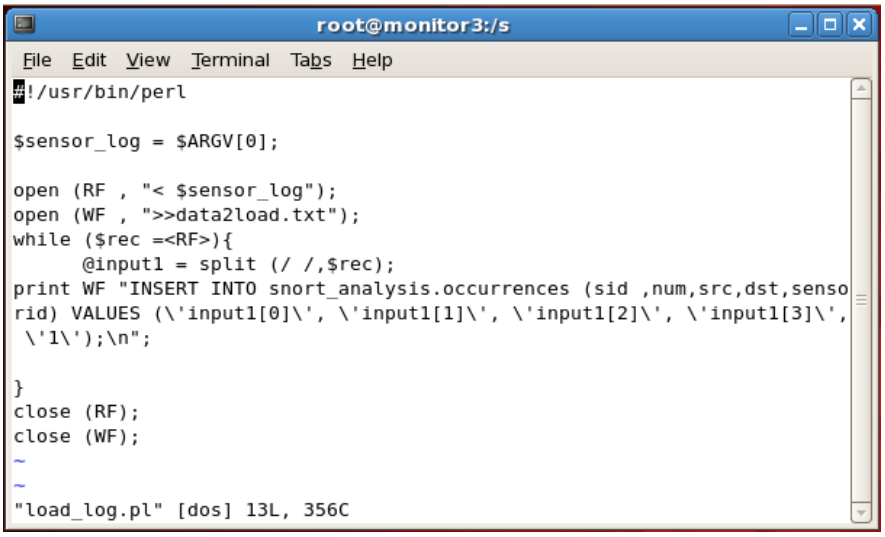

Figure 5.10: Script to Format Alert Logs of IDS Manager

Next, the following command is executed to put such formatted logs into the IDS manager's database.

```
mysql < data_directory/data2load -u ids-manger -pxxxxxx  -D snort_analysis
```

5.4.2 Alert Log Analysis

The IDS Manager analyzes alert logs of all IDS sensor in order to identify the characteristics of intrusions in the network, and determine which active rules should be on each IDS sensor. A web interface is designed for easy query processing and reporting. Figure 5.11 shows a sample query for finding the occurrences of rules category, and Figure 5.12 shows the output of rules category most occurred in intrusion traffic.

SQL query:

```
SELECT category.cname AS Rules_used, SUM( occur.num ) AS Occurence
FROM sensor
INNER JOIN (
  (
    category
    INNER JOIN ruleset ON category.cid = ruleset.cid
  )
  INNER JOIN occur ON ruleset.sid = occur.sid
) ON sensor.sensorid = occur.sensorid
WHERE (
  (
    (
      occur.sensorid
    ) =4
  )
)
GROUP BY category.cid, category.cname, sensor.sensorname
ORDER BY Occurence DESC
LIMIT 0 , 30;
```

Figure 5.11: A Sample Query Showing Occurrences Found by Rule Category

Rules_used	Occurence
snmp	109301
icmp	1500
netbios	173
sql	129
scan	22
web-misc	5
attack-responses	1

Figure 5.12: Sample Output of Occurrences Found by Rule Category

5.4.3 IDS Manager Report Interface

To facilitate the display of intrusion analysis results, the web interface is created for monitoring the status of each IDS sensor since its status is sent to the IDS manager every minute. Figure 5.13 shows the screen display for monitoring IDS sensors, and Figure 5.14 shows a sample output of intrusion logs obtained from each IDS sensor.

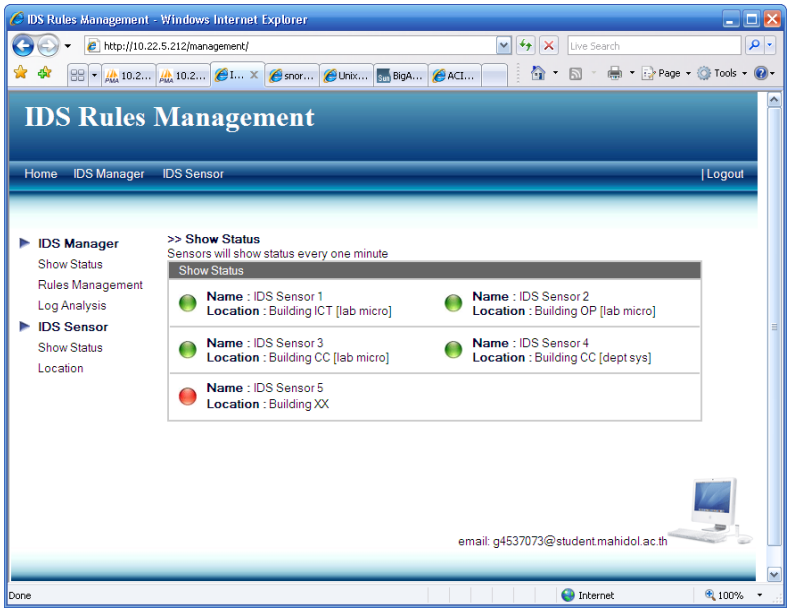

Figure 5.13 Screen Display for Monitoring IDS Sensors

Rule_Name	Occurrence	Sensor_ID	cid
SNMP public access udp	72380	4	31
SNMP request udp	72380	4	31
SNMP trap udp	7681	4	31
ICMP Destination Unreachable Communication	5674	2	11
MISC UPnP malformed advertisement	5283	1	15
ICMP Destination Unreachable Communication	2878	4	11
ICMP L3retriever Ping	2121	4	11
TFTP Get	1847	2	37
ICMP Destination Unreachable Communication	952	4	11
ICMP Destination Unreachable Communication	783	1	11
SCAN UPnP service discover attempt	759	4	34
ICMP Destination Unreachable Communication	651	3	11
TFTP Get	648	3	37
ICMP Destination Unreachable Communication	535	2	11
MS-SQL ping attempt	483	4	35
MS-SQL ping attempt	471	3	35
SNMP trap udp	268	3	31
SNMP trap udp	239	2	31
NETBIOS SMB-DS Session Setup unicode andx us	202	4	18
ICMP Destination Unreachable Communication	98	3	11
ICMP PING NMAP	62	2	11
ICMP PING NMAP	60	4	11
NETBIOS SMB Session Setup unicode username	56	4	18

Figure 5.14: Sample Output of Intrusion Logs of IDS Sensors

5.5 Rules Update

There are two ways to update Snort rules at IDS sensors. One is to specify some rules to be updated by the system administrator, and the other is to update all rules obtained from snort.org. The rules management web interface is designed for specifying only some rules to be updated at one or every IDS sensor. Figure 5.15 shows the screen display of IDS Rules Management for IDS sensors. When the IDS manager wants to update any rules at IDS sensors, it needs to put such rules in a specific directory so that a written “cron” job can download such rules to all IDS sensors and Snort is restarted. In addition, Figure 5.16 shows the script that checks the version of rules before updating them at each IDS sensor.

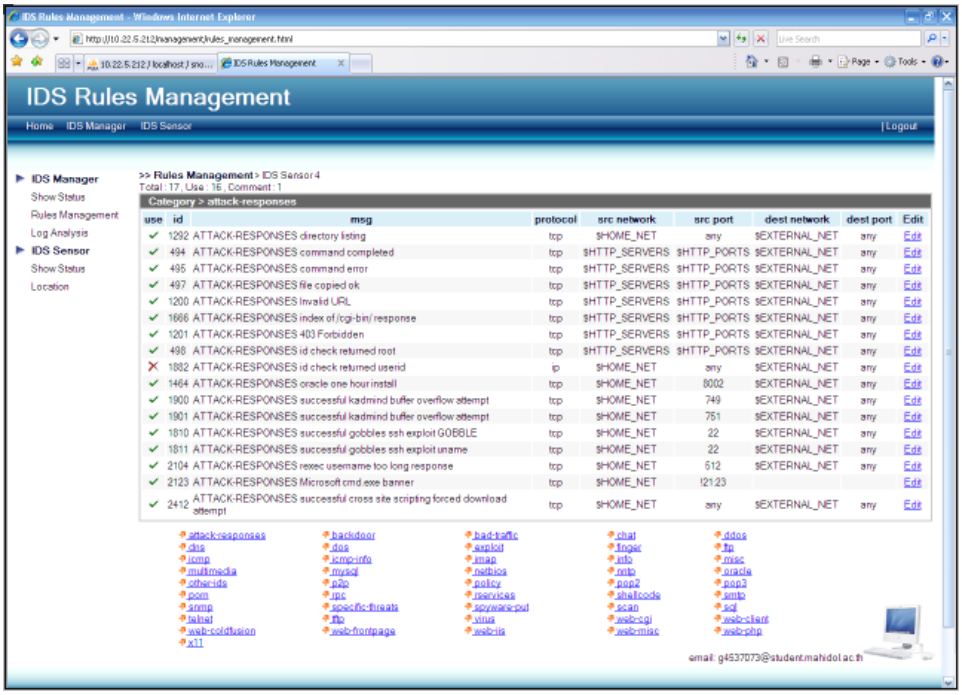

Figure 5.15: Screen Display of IDS Rules Management for IDS Sensor

5.6 False Alarm Analysis

False alarms are alarms that are not real. They could be generated by network monitoring software that collects network information periodically. Such software mostly uses SNMP protocol. However, if we found SNMP false alarms in normal traffic, it may indicate potential intrusions as well. Thus, the analysis of false alarms requires clear understanding of network behavior under inspection. In this work, false alarms could be determined from rule categories used at each IDS sensor.

Figure 5.17 shows types of intrusion protocols obtained from each IDS sensor. Obviously, SNMP has the most occurrences, and ICMP is the second most occurrences. After looking at SNMP packets in detail, we found that one particular network generates lots of monitoring traffic, and thus produces lots of SNMP packets as well as ICMP packets. When such packets are manually eliminated, the graph shown in Figure 5.18 indicates that MISC intrusion type has the most occurrences. Next, we do further analysis and found that the MISC alert type is matched with "MISC UPnP malformed advertisement Rule" shown in Figure 5.19 actively distributed within our network, and it indicates that some Windows machines have been compromised and produced such intrusion traffic to the network.

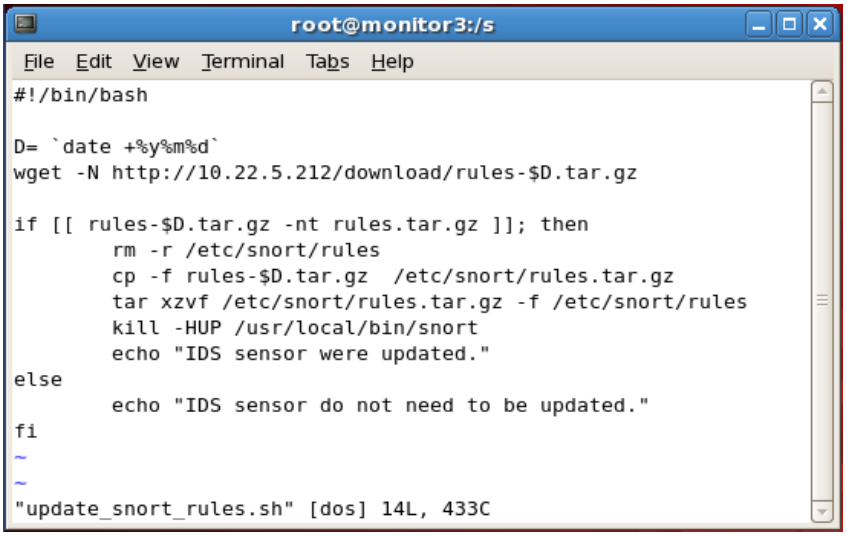

Figure 5.16: Rules Checking before Updating at IDS Sensor

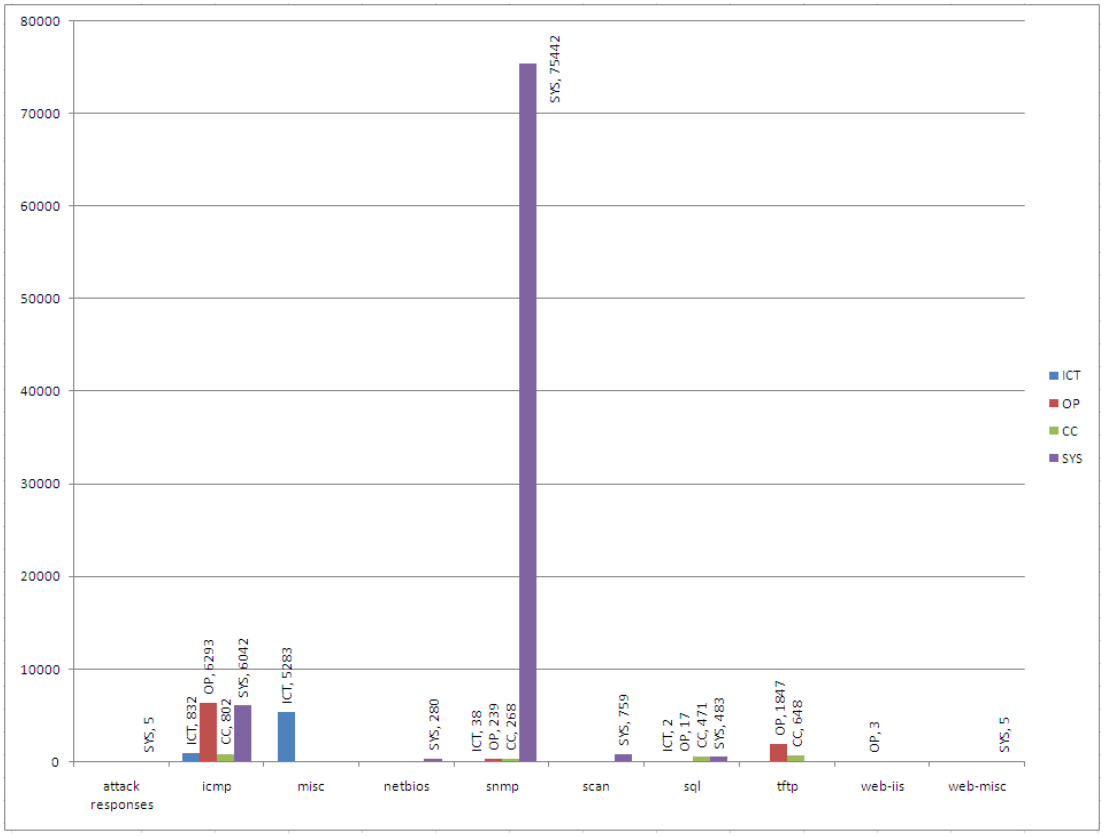

Figure 5.17: Graphs of Intrusion Types at each IDS Sensor

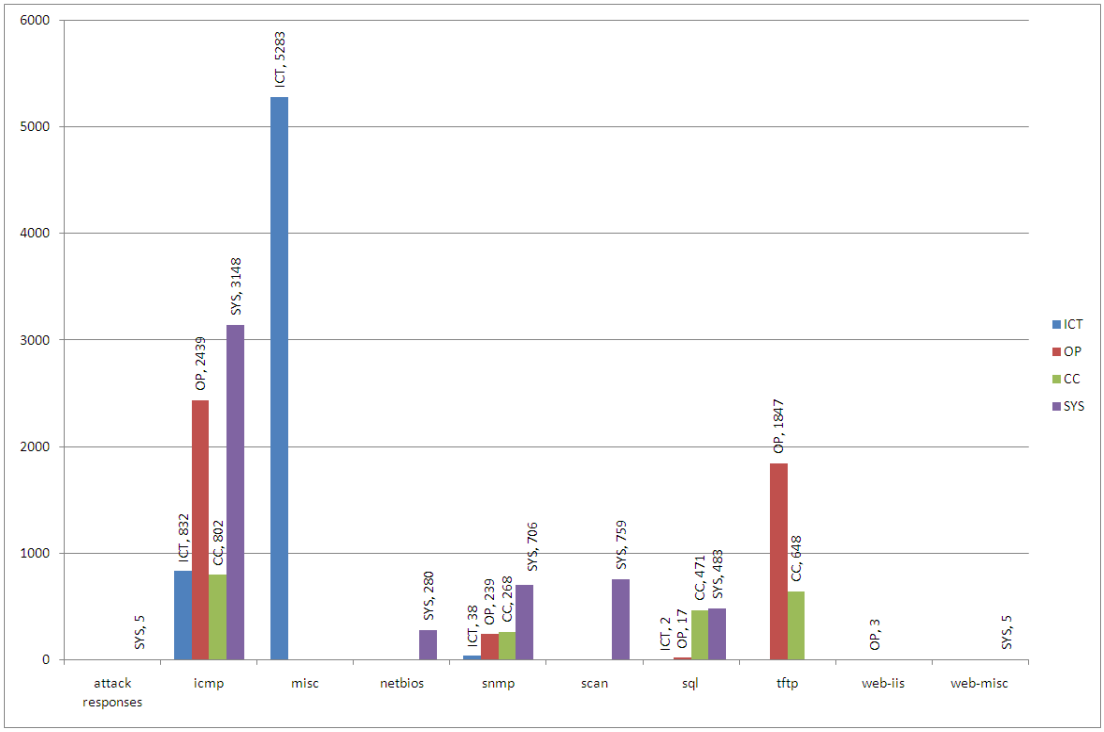

Figure 5.18: Graphs of Intrusion Types after Eliminating False Alarms

Figure 5.19: Reference of “MISC UPnP malformed advertisement rule”

5.7 Experiments

For experiments, Snort IDS is installed at four places within Mahidol University Campus Network (MUC-Net) for collecting alert logs and examining intrusion traffic from each segment of our network. The four places are our IDS sensors and they are ICT student lab, OP student lab, CC student lab, and CC system lab as shown in Figure 5.20. In this work, alert logs of intrusion information are collected from September 17 to October 19, 2007 for the total of 25 days.

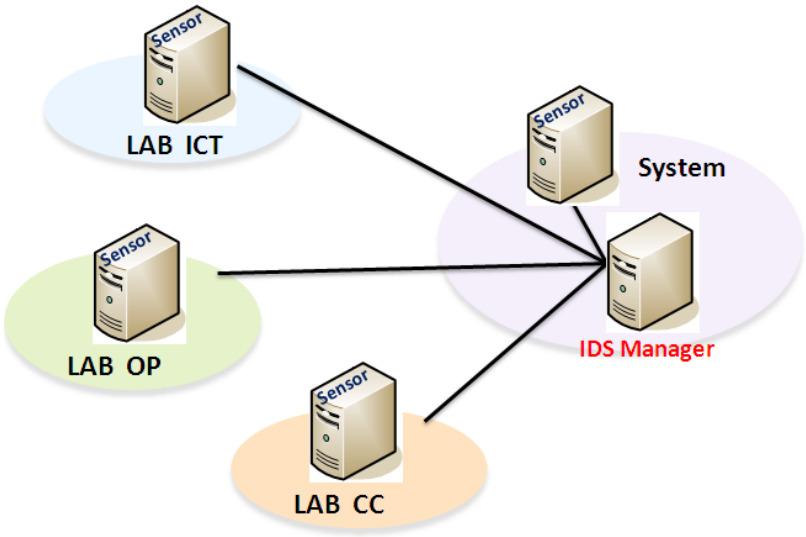

Figure 5.20: Network Configuration for Experiments

Each IDS sensor collects alert logs by capturing packets arrived, and examining their headers and contents whether they are matched with the active Snort rules defined. Those logs are recorded into the IDS sensor's database. At a specified time, the logs will be sent to the IDS manager for further analysis. The IDS manager is responsible for monitoring the status of each IDS sensor, and for analyzing the overall alert logs obtained. It also downloads the latest updates of Snort rules and distributes them to individual IDS sensors.

5.7.1 Results of IDS Sensors Analysis

IDS sensors mostly record alert logs happened from 9.00 am to 5.00 pm every day. IDS sensors examine intrusion packets with IDS rules. At every hour, IDS sensors analyze alert logs and send them to IDS manager for further analysis. From the period of our collection, the outputs shown in Figure 5.21 from four IDS sensors indicate different number of intrusion packets. Figures 5.22 to 5.25 illustrate different rules categories usage of each computer lab.

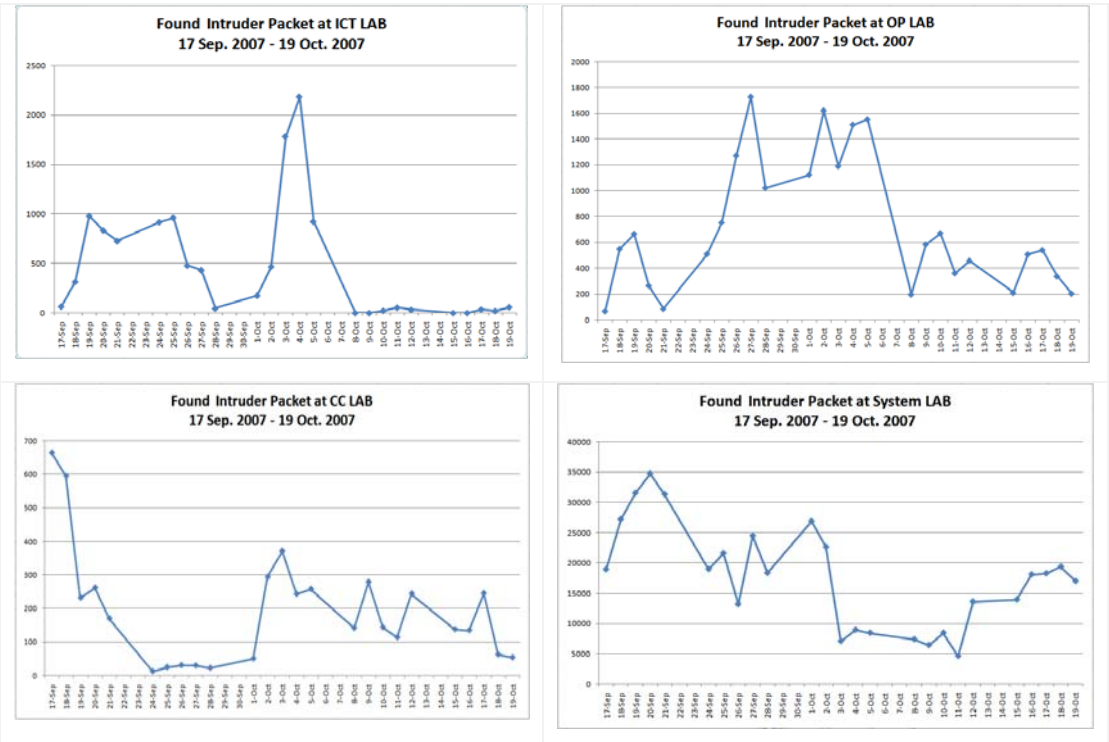

Figure 5.21: Intrusion Packets from each IDS Sensor

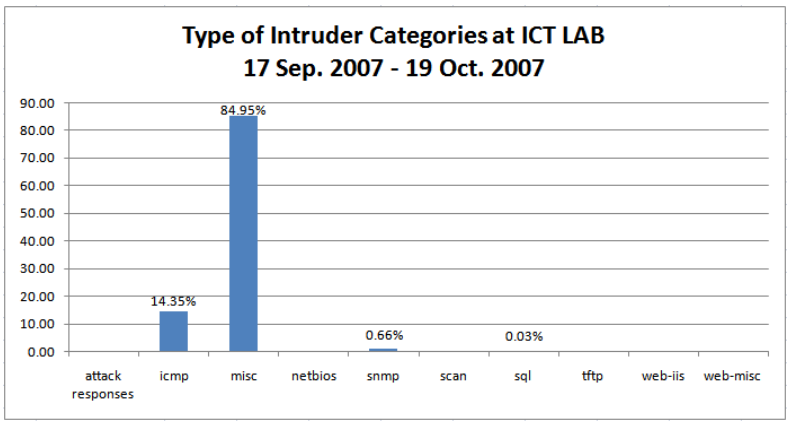

Figure 5.22: Distribution of Rules Categories at ICT Lab

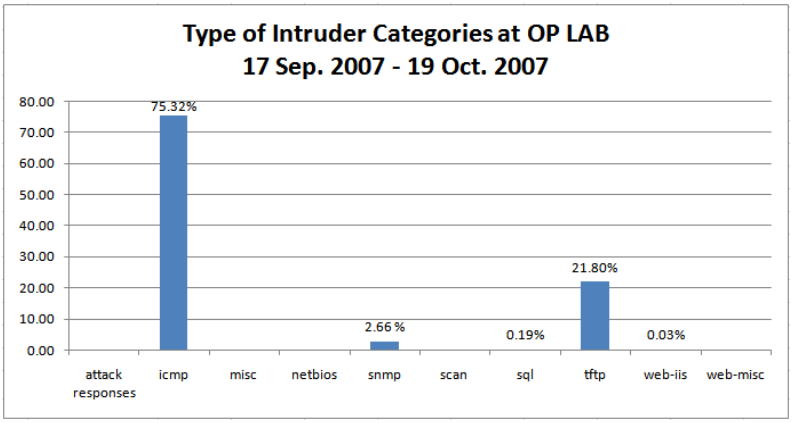

Figure 5.23: Distribution of Rules Categories at OP Lab

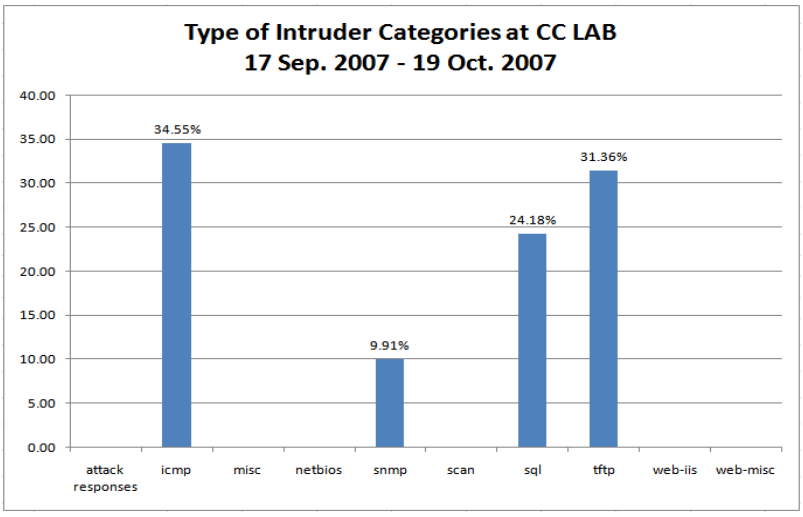

Figure 5.24: Distribution of Rules Categories at CC Lab

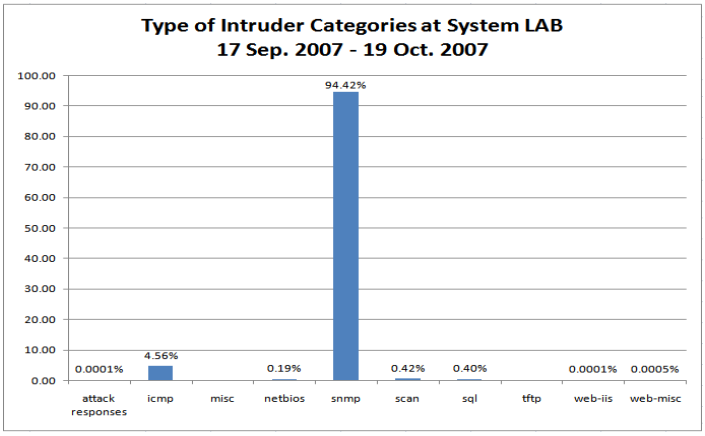

Figure 5.25: Distribution of Rules Categories at System Lab

5.7.2 Results of IDS Manager Analysis

IDS manager analyze alert logs obtained from IDS sensors. The results indicate different intrusion characteristics of each network segment. The summary of alerts according to rules categories is shown in Table 5.3.

According to the outputs shown in Figure 5.26, we found that SNMP has the highest percentage of rules categories for alerts, and the second highest and the third highest are ICMP and MISC, respectively. Our further investigation found that most SNMP and ICMP alert logs are initiated by the system or network monitoring software. The source and destination IP addresses of those monitoring machines confirm our investigation.

Category / Date	attack-responses	icmp	misc	netbios	scan	snmp	sql	tftp	web-iis
9/17/2007		924		28	18	18,590	100	72	3
9/18/2007		1,550	198	45		26,496	134	216	3
9/19/2007	1	1,547	928	53		30,412	108	320	
9/20/2007	1	1,582	830	40	4	33,191	155	248	
9/21/2007		1,316	721	27	4	29,914	196	72	
9/24/2007		1,234	917	39		18,055	78	72	5
9/25/2007		1,582	922	39		20,435	98	216	5
9/26/2007		1,671	438	54	150	12,421	59	203	
9/27/2007		2,572	433	81	719	22,648	107	71	
9/28/2007		1,796	34	37	587	16,854	97	12	
10/1/2007		1,840	176	34		25,988	107	76	
10/2/2007		2,119	467	30		22,035	129	220	
10/3/2007		2,227	1,085	10	18	6,627	114	360	
10/4/2007		2,324	1,484	32		8,480	134	432	
10/5/2007		1,374	921	11	36	8,129	79	648	
10/8/2007		857		16	9	6,651	73	144	
10/9/2007		637		12	16	6,090	98	432	
10/10/2007		597	23	11	36	8,129	63	444	
10/11/2007		635	55	15	14	4,151	58	228	
10/12/2007		1,209	32	47	14	12,631	163	216	
10/15/2007		868		32	23	13,147	122	72	
10/16/2007		1,589		37	33	16,777	145	144	1
10/17/2007		1,876	36	34	46	16,647	210	216	1
10/18/2007		1,620	20	49	65	17,652	213	144	
10/19/2007		1,417	60	43	56	15,419	148	144	

Table 5.2: Number of IDS Sensor Alerts Distribution by Categories

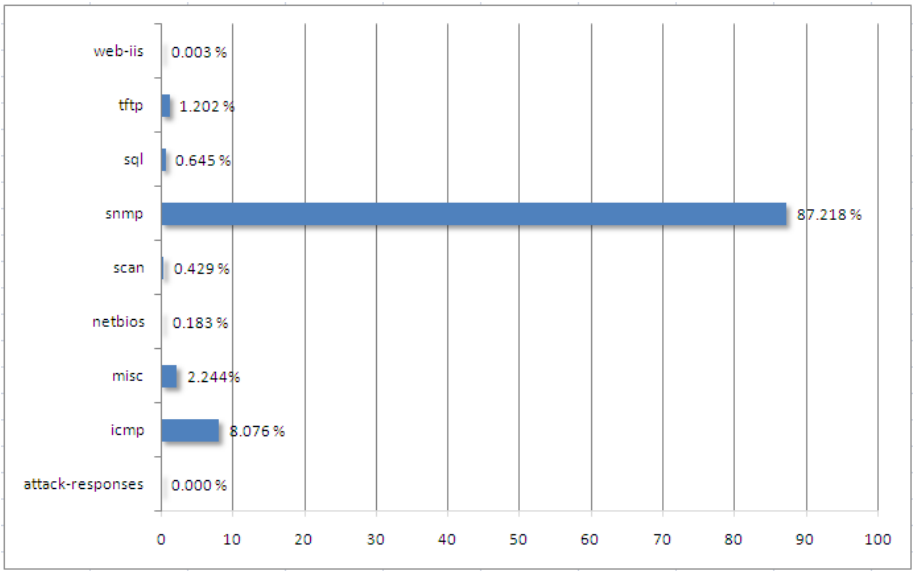

Figure 5.26: Number of Alerts Distributed by Rules Categories

The analysis of all alerts according to time of occurrences as shown in Figure 5.27 illustrates that the time having the highest number of alerts is 9 AM. But, after reducing the number of fault alarms caused by SNMP traffic, the time having the highest number of alerts is 11 AM and it is MISC traffic. The number of SNMP, ICMP, and MISC alerts is shown in Figure 5.28 to 5.30. These results imply that this is the time we have most users in the computer lab or the security level is reduced to accommodate the file sharing of some applications.

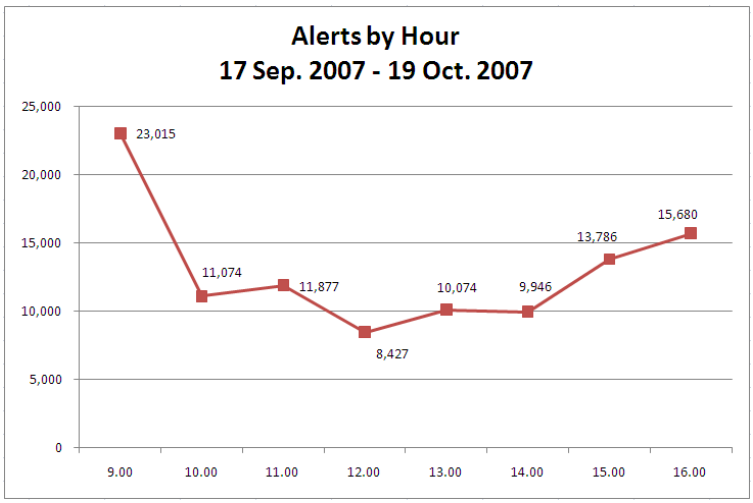

Figure 5.27: All Alerts by Hour

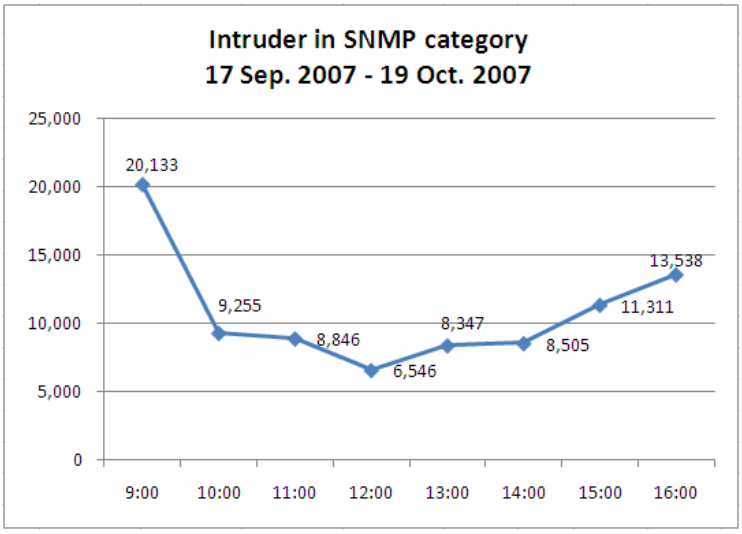

Figure 5.28: SNMP Alerts by Hour

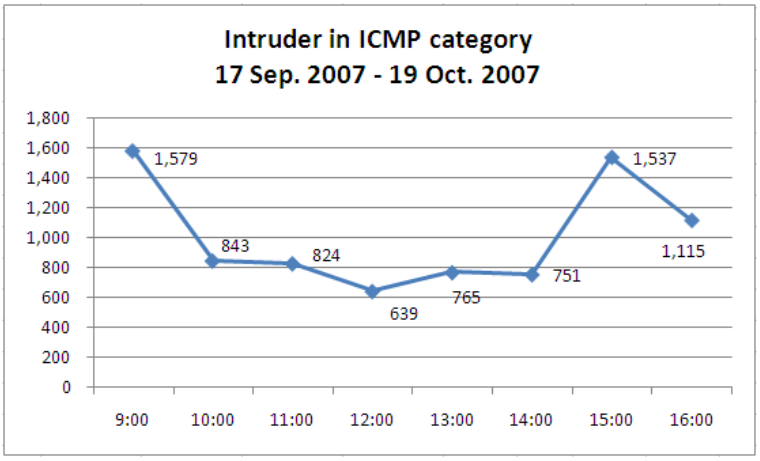

Figure 5.29: ICMP Alerts by Hour

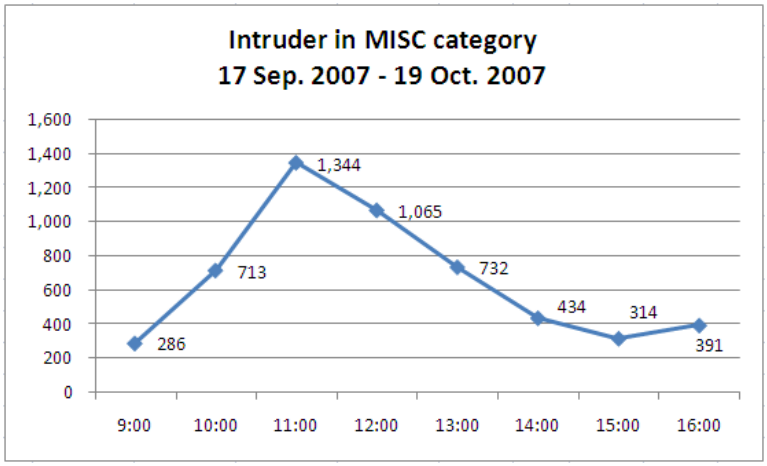

Figure 5.30: MISC Alerts by Hour

5.7.3 Update Rules

In this work, the time for updating rules from the IDS manager to all IDS sensors is specifically set. We download the new rules from Snort.org and automatically update them to all IDS sensors. The rules consistency is tested by changing some details of rules so that they could be updated again. We found that the rules of IDS sensors at four sites are the same.

5.7.4 Reports of False Alarms

From the analysis of false alarms previously presented, we found that Snort has no knowledge about operating network configuration and has no policy to define privileges of specific IP addresses for particular purposes. During the period of our collected data, we found the average of 19,018 alerts per day. After patching and eliminating false alarms, the average number of alerts from 29 November to 16 December is reduced to 266 alerts per day. In other words, the number of alerts is reduced by 98 %. The overall picture of intrusion types found at each computer lab after eliminating false alarms is shown in Figure 5.31.

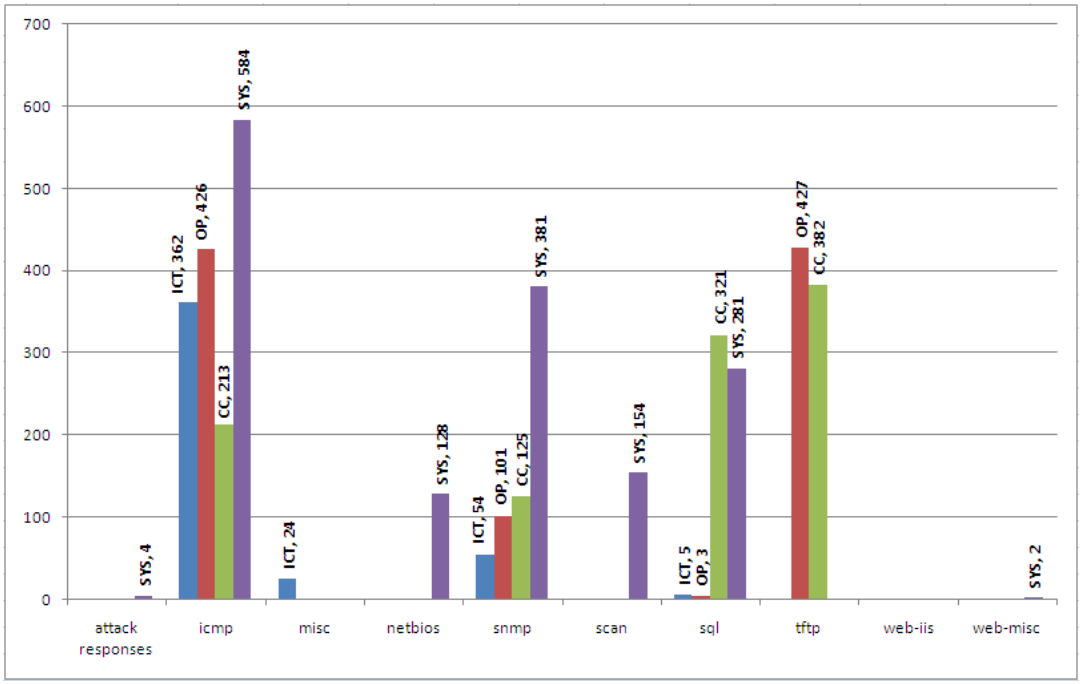

Figure 5.31: Intrusion Types after Patching and Eliminating False Alarms

CHAPTER 6
DISCUSSION AND CONCLUSION

This chapter summarizes our proposed work, discusses the experimental results, concludes our contribution, and suggests some future work.

6.1 Summary of the Proposed Work

The goal of this thesis is to propose a simple and flexible architecture for managing multiple IDS sensors distributed across campuses of Mahidol University. The system architecture consists of two components: IDS sensors and the IDS manager. IDS sensors are installed at many sites, and responsible for collecting alert logs before sending the preliminary log analysis and rules usage to the IDS manager. The IDS manager will collect alert logs from several IDS sensors and perform the analysis to find out what could be potential threats within Mahidol University network. It is also responsible for ensuring the consistency of rules updates across IDS sensors. The IDS manager also monitors the number of false alarms as they would affect the IDS performance. Thus, the system consists of several modules: log collection, log parser, log analysis and Snort reconfiguration.

In the implementation, Snort is used as an IDS sensor and MySQL is used as the database for storing logs and performing log analysis. In addition, shell scripts such as Perl scripts are used for log parser and for communicating between IDS sensors and the IDS manager. PHP language is also used for creating the web interface.

For experiments, four Snort machines are installed at four computer labs. The alert logs are collected every day from September 17 to October 22, 2007, and the interesting outputs are presented in the previous chapter.

6.2 Discussion of Experimental Results.

We discuss the experimental results and make some observations as follows.

- Each network has its own nature of usage traffic. Thus, each IDS sensor detects different pattern of intrusions, and should have different active rules installed.
- For the analysis of network traffic and false alarms, network information such as IP address for monitoring is needed to support the analyzed results.
- The time synchronization of data on each IDS sensor is very important for the analysis. If the time at each location is missing, we may obtain incorrect information from the analyzed data. In our experiments, many sites are used to collect information. Thus, a clock server is set for serving this requirement.
- Differences in date interval for collecting data may show different information since data are initially collected at the period of final exams for students. Thus, typical data collected may be less than those at the regular time. But, the trends are still the same.
- Snort is a network IDS. Hence, it can monitor and detect intrusion traffic only at one segment of a network. It cannot detect intrusions of the whole campus. Snort could be put at the main network, but the traffic volume would be very high, and typically Snort still has the problem with the performance. Thus, we must find a solution to solve this if we want to really deploy Snort to the full scale.
- Typically, Snort itself could not detect new intrusion patterns. As a result, the system administrator must monitor the log analysis, and take appropriate actions by himself.
- Currently, most intrusions are the results of vulnerabilities of Microsoft Windows operating system. If the security patches are frequently updated, such vulnerabilities could be ignored and disregarded by IDS sensor.

6.3 Suggestions for Future Work

Our proposed work introduces some open issues in IDS research area, and here is the list for improvements.

- High security is required for the communication mechanism between IDS sensor and IDS manager. For example, a technique for authenticating IDS sensors and the IDS manager would be necessary and should communicate over SSL. Most security or IDS products use agents and specific ports for communication, but open source software like Snort could use the standard ports like HTTP, HTTPS, FTP and FTPS.
- Alert logs from IDS sensors are very large. Thus, an effective way to analyze such logs would enhance the IDS performance.
- Exchanging rules among several IDS via IDMEF format would allow the system to be flexible and interoperable with the standards.
- The technique for analyzing and reporting false alarms automatically is necessary.
- Additional technique for monitoring IDS sensor such as mobile agents would make the system effective.
- Additional data mining technique for analyzing alert logs collected at the IDS manager would help us identify potential threats.

REFERENCES

1. Activeworx. Available at: http://www.activeworx.org
2. Bro. Available at: http://bro-ids.org
3. Honeynet. Available at: http://www.honeynet.org
4. Libidmef. Available at: http://sourceforge.net/projects/libidmef
5. Libpcap. Available at: http://tcpdump.org and http://www.winpcap.org
6. Libxml. Available at: http://xmlsoft.org and http://libxmlplusplus.sourceforge.net
7. Mysql. Available at: http://www.mysql.com
8. Ntop. Available at: http://www.ntop.org
9. Pcre. Available at: http://www.pcre.org
10. Phpmyadmin. Available at: http://www.phpmyadmin.net
11. Snort. Available at: http://www.snort.org
12. Tcpdump. Available at: http://www.tcpdump.org
13. Caswell B. **Snort 2.0 Intrusion Detection**. Syngress Publishing Inc., Rockland, 2003.
14. Debar H, Curry D, Feinstein B. **The Intrusion Detection Message Exchange Format**. Internet Draft, IETF, March 2007. http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-17.txt
15. Deri L, Carbone R, Suin S. **Monitoring Networks Using Ntop**. Proceeding of the 2001 IEEE/IFIP International Symposium on Integrated Network Management. May 14-18; 2001. pp.199-212.
16. DuBois Paul. **MySQL: The Definitive Guide to Using, Programming, and Administering MySQL 4 Databases,** 2nd Edition. Sams Publishing, 2003.
17. Endorf C, Schultz E, and Mellander J, **Intrusion Detection and Prevention**, McGraw-Hill, 2004. .

18. Gabriel J and Mark D., **Alarm Correlation**. IEEE Network. November 1993. pp.52-59.

19. Jakobson G and Weissman M, **A New Approach to Message Processing in Distributed TMN.** Proceedings of the Fourth IFIP/IEEE International Workshop on Distributed Systems, October 5-6, 1993.

20. Jones A and Song L. **Temporal Signatures for Intrusion Detection**, Proceedings of 17th Annual Computer Security Applications Conference ACSAC, Dec 10-14; 2001. p. 252-261.

21. Kreibich C and Sommer R, **Policy-controlled event management for distributed intrusion detection**, Proceedings of 25th IEEE International Conference of Distributed Computing Systems, June 6-10, 2005. p.385-391.

22. Levine J, Grizzard J and Owen H. **Using Honeynets to Protect Large Enterprise Networks**. Proceedings of IEEE Security and Privacy, November 2004. p. 73-75.

23. Martin Roesch. **Snort – Lightweight Intrusion Detection for Networks**. In Proceedings of 13th Systems Administration Conference (LISA), November 7-12; Seattle, Washington 1999. p. 229-238.

24. Mathieu B, Laurent O, and Vincent G. **Global Intrusion Detection: Prelude Hybrid IDS.** Villeurbanne, France, 2003

25. Melanie Middlemiss. **Framework for Intrusion Detection Inspired by the Immune System**. Department of Information Science, University of Otago, July 2005.

26. Orwant J. **Perl 5 Interactive Course**, Waite Group, 1996.

27. Richard A. **Multi-sensor agent-based intrusion detection system**. Proceedings of the 2nd Annual Conference on Information Security, 2005. p.100 – 103.

28. Rubin S, Jha S, Miller BP. **Automatic generation and analysis of NIDS attacks**. Proceedings of 20th Annual Computer Security Applications Conference (ACSAC), December 6-10; 2004. p. 28-38.

29. Russ M. **Activeworx IDS Policy Manager 2.0: Rules management for multiple sensors**. ISSA Journal; January 2007. p. 40-43.

30. Schaelicke L, Slabach T, Moore B, and Freeland C. **Characterizing the Performance of Network Intrusion Detection Sensors**. Proceedings of the 6th International Symposium on Recent Advances in Intrusion Detection (RAID), 2003.

31. Todd Meadors. **Linux Shell Script Programming**. Thomson-Paraninfo, 2007.

APPENDIX

APPENDIX A
INSTALLATION and CONFIGURATION of SNORT IDS SENSOR

This appendix shows the steps for installing Snort used in the implementation of IDS sensors. The version of Snort used runs on Linux operating system which is Red Hat Enterprise Linux 5 and Fedora core 5. The prerequisite for setting up is Mysql server used to store logs of IDS sensors when suspicious packets in the network are found.

A.1 IDS Sensor Installation

IDS sensor used in this work is Snort version 2.4.5 and has the rules snapshot version 2.4. The IDS sensor can generate alerts in IDMEF format as well. In addition, Snort needs the following applications.

- *Libpcap* : Recommend libpcap version 0.8.1
- *Pcre* : Recommend Pcre version 7.2
- *Libxml* : Recommend Libxml2 vesion 2.6.29
- *Libidmef* : Recommend Libidmef version 1.0.2-beta1 and must to use specific directory to /usr

The step of installlation are as follows.

- *tar xzf snort-x.x.x.tar.gz -C /usr/local*
- *tar xzf snort-idmef-plugin-2.0.0beta.tar.gz -C /usr/local*
- *cd /usr/local*
- *ln -s snort-2.4.5 snort*
- *cd snort-idmef*
- Edit */snort-idmef/install-demef-plugin.sh*
  snortdir =”/usr/local/snort”
  tardir=”/usr/local/snort-idemf”
- Run *./install-demef-plugin.sh*
- *cd /usr/local/snort*
- Run *autogen.sh*

- *./configure –enable-idmef *
  *–with-mysql *
  *–with-libxml2-includes=/usr/local/include/libxml2 *
  *–with-libxml2-libraries=/usr/local/lib *
  *–with-libidmef-includes=/usr/local/include *
  –with-libidmef-libraries=/usr/local/lib
- *make*
- *make install*
- Edit *schemas/create_mysql*
  add the first Line : *USE snort;*
- *mysql < schemas/create_mysql -u root -p -D snort*
- *mkdir /etc/snort*
- *mkdir /etc/snort/rules/*
- *cp snortrules-pr-2.4.tar.gz /etc/snort/rules*
- *cd /etc/snort/rules*
- *tar -xzvf snortrules-pr-2.4.tar.gz*
- *cp rules/* .*
- *cp /usr/local/snort-idmef/appen_idmef.pl /etc/snort/rules*
- *cp /usr/local/snort-idmef/idmefify.sh /etc/snort/rules*
- Edit */etc/snort/rules/idmefify.sh*
  change this line *"/projects/event-correlation/devel/idmef-plugin/append_idmef.pl"*
  to "*/etc/snort/rules/append_idmef.pl "*
- *cd /etc/snort/rules/*
- *./idmefify.sh*
- *cp /usr/local/snort-idmef/*.dtd /etc/snort*
- *cp /usr/local/snort/etc/snort.conf /etc/snort*
- *cp /usr/local/snort/etc/*.config /etc/snor*
- *cp /usr/local/snort/etc/*.map /etc/snort*
- Modify *snort.conf*
  Add directory for snort rules after *RULE_PATH* and add */etc/snor/rules*
- *mkdir /var/log/snort*
- *chmod 700 /var/log/snort*

A.2 IDS Sensor Configuration

After installing, Snort must be properly configured. The file is snort.conf in the directory /etc/snort/. The local network environment such as network variables, rule directory, environment, and outputs are configured. The example to configure Snort in snort.conf is shown below.

```
var HOME_NET   10.22.5.0/24
var EXTERNAL_NET    any
var RULE_PATH  /etc/snort/rules
include  /etc/snort/classification.config
include  /etc/snort/reference.config

include $RULE_PATH/local.rules
include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/exploit.rules
include $RULE_PATH/scan.rules
include $RULE_PATH/finger.rules
include $RULE_PATH/ftp.rules
include $RULE_PATH/telnet.rules
include $RULE_PATH/rpc.rules
include $RULE_PATH/rservices.rules
include $RULE_PATH/dos.rules
include $RULE_PATH/ddos.rules
include $RULE_PATH/dns.rules
include $RULE_PATH/tftp.rules

include $RULE_PATH/web-cgi.rules
include $RULE_PATH/web-coldfusion.rules
include $RULE_PATH/web-iis.rules
include $RULE_PATH/web-frontpage.rules
include $RULE_PATH/web-misc.rules
include $RULE_PATH/web-client.rules
include $RULE_PATH/web-php.rules

include $RULE_PATH/sql.rules
include $RULE_PATH/x11.rules
include $RULE_PATH/icmp.rules
include $RULE_PATH/netbios.rules
include $RULE_PATH/misc.rules
include $RULE_PATH/attack-responses.rules
include $RULE_PATH/oracle.rules
include $RULE_PATH/mysql.rules
include $RULE_PATH/snmp.rules
include $RULE_PATH/smtp.rules
include $RULE_PATH/imap.rules
include $RULE_PATH/pop2.rules
```

```
include $RULE_PATH/pop3.rules
include $RULE_PATH/nntp.rules
include $RULE_PATH/other-ids.rules

include $RULE_PATH/web-attacks.rules
include $RULE_PATH/backdoor.rules
include $RULE_PATH/shellcode.rules
include $RULE_PATH/policy.rules
include $RULE_PATH/porn.rules
include $RULE_PATH/info.rules
include $RULE_PATH/icmp-info.rules
include $RULE_PATH/virus.rules
include $RULE_PATH/chat.rules
include $RULE_PATH/multimedia.rules
include $RULE_PATH/p2p.rules
include $RULE_PATH/experimental.rules

output idmef: $HOME_NET output=alert dtd=/etc/snort/idmef-message.dtd \
analyzerid=IDS1 indent=true facility_default=file|/var/log/snort/idmef-messages.log \
alert_id=/var/log/snort/alert_id_num default=ascii
output database: log, mysql, user=snort password=snort dbname=snort host=localhost
```

BIOGRAPHY

NAME	Miss Chitthaporn Sae-Ngow
DATE OF BIRTH	19 November 1976
PLACE OF BIRTH	Bangkok, Thailand
INSTITUTIONS ATTENDED	Mahidol University, 1998-1999: Bachelor of Science (Computer Science) Mahidol University, 2002-2007: Master of Science (Computer Science)
POSITION & OFFICE	1998 - Present, Mahidol University Computing Center, Rama VI Road, Rajathewi, Bangkok 10400, Thailand. Position : Computer Technical Officer Tel. 0-2354-4333 E-mail : cccso@mahidol.ac.th