Study on Intrusion Detection System for a Campus Network

by

Mazlina Shakirah Binti Zainal Abidin

Dissertation submitted in partial fulfillment of
the requirements for the
Bachelor of Technology (Hons)
(Information Technology)

JUNE 2005

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

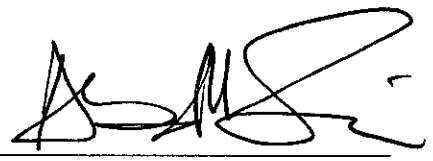
Study on Intrusion Detection System for a Campus Network

by

Mazlina Shakirah Binti Zainal Abidin

A project dissertation submitted to the
Information Technology Programme
Universiti Teknologi PETRONAS
In partial fulfillment of the requirement for the
BACHELOR OF TECHNOLOGY (Hons)
(INFORMATION TECHNOLOGY)

Approved by,



(Mr. Abdullah Sani b. Abd Rahman)

UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

JUNE 2005

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

MAZLINA SHAKIRAH BINTI ZAINAL ABIDIN

TABLE OF CONTENTS

ABSTRACT

All final year students in UTP are required to undertake a final year project (FYP) paper, which are a design and/or research-based subject. It requires student to do research; design and/or development work in each discipline, especially on real-world problems which would motivate student to produce practical solutions. This project title is "Study on Intrusion Detection System for a Campus Network". It is a research and development work project. The objective of the project is to make sure student do a research in the area that relevant with specified title. Beside, student also needs to make a test bed application that is used in implementing the IDS. This project scope will focus on implementing the IDS in campus network and how to simulate the attacks besides measure it effectiveness in detecting any intrusion.

ACKNOWLEDGEMENT

First and foremost, thank God for this opportunity and seeing me through some truly difficult times especially through the journey of completing this project. I would like to express my warmest gratitude and appreciation to all parties who have contributed towards the success of this Final Year Course Project.

I would like to express my appreciation especially to the following people:

- Mr. Abdullah Sani Abd. Rahman
  Supervisor
- Mr. Mohammed Noor Ibrahim
  IT/IS Final Year Project Coordinator
- Mrs. Vivian Yong Suet Peng
  IT/IS Final Year Project Coordinator

Also my utmost gratitude goes to my family who help in many ways. Only Allah may repay them.

This special thanks and appreciation also dedicated to all my colleagues and my friends for their continued support, guidance and contribution to the success of this Final Year Project. With the full cooperation from the various people above, I have successfully achieved the objective of this project.

Thank You.

LIST OF FIGURES AND TABLES

ABBREVIATION AND NOMENCLATURES

1. IDS	Intrusion Detection System
2. NIDS	Network Intrusion Detection System
3. TCP/IP	Transmission Control Protocol/Internet Protocol
4. DNS	Domain Name System
5. IP	Internet Protocol
6. OSI	Open Systems Interconnection
7. PC	Personal Computer
8. IT	Information Technology
9. ARP	Address Routing Protocol
10. DoS	Denial of Services
11. UDP	Unit Data Protocol
12. ICMP	Internet Control Message Protocol

ABOUT THIS DISSERTATION

Chapter 1 – Introduction

Consists the basic information of the project, comprises of its background, its problem statement, its objectives and the scope involved. This section also described the project overview and the significant of this project.

Chapter 2 – Literature Review

This chapter contains the acknowledged findings on this field, consisting of relevant theories, hypothesis, facts and data which are relevant to the objective and the research of this project.

Chapter 3 – Methodology and Tools Used

This chapter features the detailed description of methodology and procedure of completing this project. This methodology is implemented in order to ensure that the project is running as required. The tools that is used for this project also is discussed in this chapter.

Chapter 4 – Results and Discussion

This chapter reports the product details and findings, which supports the project work. The main discussion and finding here is a discussion about the result obtained and the intrusion detection system effectiveness.

Chapter 5 – Conclusion and Recommendation

This chapter briefly states about the project done and a few recommendations for the future enhancement.

CHAPTER 1
INTRODUCTION

Computer and networking technologies dominate much of our lives until today. Many of us rely on these technologies everyday which all our works and communications are enabled by these systems. Even as we rely on these systems, we're painfully aware of the flaws and imperfections in them.

In this network world, the needs for security and appropriate systems of control are clear and the marching orders for those who would secure computer systems and networks are ambitious indeed. The security achieved must be reasonable and balance with the needs for privacy. It must be flexible enough to accommodate a global range of statues and regulations besides consistent enough to track the criminal across multiple jurisdictions.

The blend of management and technical measures is necessary to meet these security requirements, which is very complex. So, the area of audit and intrusion detection has become an important part of computer and network security. The functions provided by this technology serve the goals of security by providing trace back and detection capabilities and also by monitoring the health and trustworthiness of other security mechanism in the system.

1.1 Background of study

Intrusion detection is the process of monitoring the events occurring in a computer system or network. It also inspects all inbound and outbound network activity and identifies suspicious patterns that may indicate a network or system attack from someone attempting to break into or compromise a system. This technology is designed to monitor computer activities for the purpose of finding security violations which is varies from one organization to another organization. Intrusion detection

system spots malicious activity as it occurs and responds or alert the user appropriately. Unfortunately accomplishing this full goal of intrusion detection is not yet possible because networks are complex and the traffic and activities are diverse besides most of the attacks are almost found daily. This situation requires the system to be updated frequently.

There are several ways to categorize an IDS:

1. Misuse Detection vs. Anomaly Detection

    - In misuse detection, the IDS analyze the information it gathers and compares it to large databases of attack signatures. Essentially, IDS is looking for a specific attack that has already been documented. Like a virus detection system, misuse detection software is only as good as the database of attack signatures that it uses to compare packets against.
    - In anomaly detection, the system administrator defines the baseline, or normal, state of the network's traffic load, breakdown, protocol, and typical packet size. The anomaly detector monitors network segments to compare their state to the normal baseline and look for anomalies.

2. Network-based vs. Host-based systems

    - In a network-based system, or NIDS, the individual packets flowing through a network are analyzed. The NIDS can detect malicious packets that are designed to be overlooked by a firewall's simplistic filtering rules.
    - In a host-based system, the IDS examines at the activity on each individual computer or host.

3. Passive System vs. Reactive System

    - In a passive system, the IDS detect a potential security breach, log the information and signal an alert.
    - In a reactive system, the IDS respond to the suspicious activity by logging off a user or by reprogramming the firewall to block network traffic from the suspected malicious source.

IDS attempts to detect unauthorized or malicious activities in a network or on a host system based on signature based or anomaly based. Signature based allow IDS to look for patterns that are known to be intrusive in packets or audit logs and anomaly based allows ids to look for abnormal activity usually requires a template of normal activity.

Though they both relate to network security, IDS is differs from a firewall in that a firewall looks out for intrusions in order to stop them from happening but the IDS is only detect them but do not stop them. The firewall limits the access between networks in order to prevent intrusion and does not signal an attack from inside the network while IDS evaluates a suspected intrusion once it has taken place and signals an alarm. IDS also watch for attacks that originate from within a system.

1.2 Problem Statement

1.2.1 Problem Identification

There were many complaints from the student about the performance of the network. Most of them say that the network performance is slow and it affects their works which require them to surf from the Internet. Network performance can become slower when the transition rate is low or the connection bandwidth is less. This is happen because many packets is transmitted in one time and each packet need to be analyze before it is send to the other party. During the packet transmission, some of the packet sent may be dropped if the packet analyzer found some unknown information in the packet. Then, there also some of the packet received doesn't have the information required and this packet may send attacks to the network and it may affect other data that is being transmitted.

In the network environment, there were various type of attacks occurs which will affect the network performance such as malicious code, password crack, spoofing, brute force attack, main-in-middle and others. Most of these attacks commonly come from trusted network because the firewall that they used is only limit the access between networks not the network inside. For example IP spoofing where the

intruder sends message to a computer with an IP address indicating that the message is coming from the trusted network while the truth is not.

Nowadays there are many unknown attacks found in the network especially for malicious code type. Most of attacks from this type are found everyday as new comers where the detection system doesn't recognize it. These attacks may spread out to other network area, which will be infected. Campus network currently used firewall and also the anti virus software to patch any malicious code or other type of attacks in the network. Even though they use firewall or antivirus for their computers, the attack still can go into their computer because firewall only limit the access between networks not the network inside.

1.2.2 Significant of the project

There were many types of attacks in the network nowadays that may cause our system or our network not performs well. As there were many types of unknown attacks, the writer has to come out with own rules that can detect any intrusion or attacks that intrude trusted network. The rules are only detecting the unknown packet but not prevent it from getting into the network.

From this research perhaps it will enhance the network effectiveness and minimize the attackers' possibility to attack the computer or systems. It also hoped that the writer could produce useful rules to prevent an intrusion in the network.

1.3 Objectives and Scope of Study

1.3.1 The objectives

- To identify the type of attacks that occurs in campus network
- To provide rules that is used for intrusion detection system
- Produce a test bed environment to simulate the attacks
- To measure the effectiveness of IDS

1.3.2 Scope of study

The scope of study for this research project tends to focus on studying the intrusion detection system concept and how it can manage the network management system. Then, it also focuses on developing a framework that can improve the system to detect the unknown attacks that intrude the network.

CHAPTER 2
LITERATURE REVIEW

Many IT organizations especially in tough economic times are trying to get more out of what they already have and are asking vendors for more security functionality of existing products that are already installed in their networks. Intrusion detection is a type of network security that detects, identifies and isolates attempts to intrude or make inappropriate use of computers.

Network intrusion can be simply broken into three areas which are Before the Attack, During the Attack and After the Attack. It is very easy to break it down this way but very hard to combat how someone might gain access to our network. If the intrusions are performed for the purpose of gathering information or resource usage rather than denial of service, the attacker will do everything in his or her power to ensure that their presence is difficult to detect. This makes it even harder to detect and trace what is happening within the network environment.

2.1 Before the Attack

Most of the companies work hard to take pro-active steps to prevent network intrusions but they still happen and they still cause a lot of damage. However, if an adversary spends enough investigatory time and has the appropriate technical knowledge, they could get in if they are willing to work hard enough. As stated previously, most of the network intrusions happen from inside the trusted network completely bypassing the firewall. Most companies utilize a network based intrusion detection system which is a distributed probe that monitors internal network segments and looks for unauthorized traffic or requests.

A network intrusion detection system (NIDS) helps identify the fact that attacks may be occurring. It is designed to detect, monitor and log potential security breaches.

Current network IDS products use a predominantly passive approach to collecting data via protocol analysis garnered by watching traffic on the network. Each one monitors the traffic on specific network segments. It gets copies of its segment's traffic to inspect by "listening on promiscuous mode" and having its network interface card bring in a copy of every packet it sees. It examines this packet and attempts to determine whether they represent an intrusion attempt by comparing it to a list of known attack signatures. It does this by determining if the contents of the packet contain the signature of a known attack method that is whether it contains a string of characters that matches a specified pattern or otherwise fits rules that define known attack methods.

2.2 During an Attack

When someone is trying to attack or penetrate your network, it is commonly understood that they are trying to access the data that is not for public. They try to change or delete the company information, illegally use the server memory for their own purposes and attempting to flood a particular device rendering it unusable to be authorized users thereby creating a denial service attack.

Denial service attack is the common attack that can cause serious downtime. It looks to send a tremendous amount of traffic via bogus server requests to your critical servers. Your machine becomes overburdened and will be unable to respond to normal requests. This is a very common attack to websites or key file servers. If your machine is compromised, your machine's resources can be used in a denial of service attack for another target unbeknownst to you.

2.3 After the Attack

If an attack on your network occur, it is essential to do everything possible to ensure that it can be detected and the traffic can be captured and analyzed. After you identify that you are under attack or have been attacked, you cannot go back and collect necessary raw network traffic to piece together what happened. The only avenue at this point is to access log files of the critical machine hit in the attack, provided you had the highest detailed logging turned on. Even with this information,

you could still be missing critical data. It is like trying to complete a jigsaw puzzle without knowing the number of pieces available and how many are you missing. You may not be able to understand what the picture even looks like.

2.4 Security Test Areas

A security test is performed with two types of attack which are passive attack and intrusive attack. A passive attack is often a form of data collection which does not directly influence or trespass upon the target system or network. An intrusive attack however does not trespass upon target system or network and can be logged and used to alarm the target system or network. The process of a security test concentrates on evaluating the visibility, access, trust and alarm. All of this area is concerned because it may give bad effect on network performance where it may cause the network traffic become jammed.

2.5 Packet Sniffing

Sniffing program has been around for a long time in two forms which are commercial packet sniffers and underground packet sniffers. Each of them is used to help maintaining the networks and also is used to break into computers. In theory, it is impossible to detect sniffing programs because they are passive. They only collect packets but do not transmit anything. However, in practice it is sometimes possible to detect sniffing programs. A stand alone packet sniffer doesn't transmit any packet but when installed non-stand alone in a normal computer, the sniffing program will often generate traffic such as it might send out DNS reverse lookups in order to find names associated with IP address.

2.6 Brief Description of SNORT

Snort is the product of Marty Roesch's mission to create an open source, lightweight NIDS. His product become wildly popular and now is deployed at countless organizations from small homes to large universities. While not designed specifically for a high-bandwidth environment, Snort performs well even at high traffic ingestion rates because it uses a simple but fast decoder and detection engine.

The basic Snort architecture is made up of three main parts which are the packet decoder, detection engine and the alerting and logging system. The packet decoder is based on libpcap and can collect TCP/IP traffic at a blinding rate. The packet data is decoded layer by layer up trough the OSI model until it is passed to the detection engine.

Before the engine can compare any of the signatures in its database to the packets, the packets data is passed trough a number of user configurable preprocessors. Then, if any signature is match with the signature in database, the third part of Snort which is alert/log system will take the action prescribed. If configured, Snort will also capture the packet data relating to the alert and store it on the hard drive. The alert system will publish alerts to an area on the file system for the intrusion analyst to examine or to a remote analysis through standard remote log formats like syslog or smb messages.

Overall, all the literature review that I read before tell me about the definition, concept, analysis schemes and how to understand the network traffic which is very complicated and difficult to understand. It may take much time in order to understand the traffic clearly.

CHAPTER 3
METHODOLOGY AND PROJECT WORK

Project methodology refers to the framework that is used to structure, plan and control the process of intrusion detection system in the campus network. Creating the rules for intrusion detection system involve reusability with open source technology however limited knowledge about the existed open source acquire me to have further reference about its usability and its functions. Lack of technical capability and interdependencies of various programming language and technology requires ongoing internal testing and peer review of the different module compliant to project scope.

In completing this project, I have used five steps or five phases that helps me to plan and control the intrusion detection system. As to start my project, I had reviewed about intrusion detection system in order to give me a clear view about intrusion detection system and to help me narrow down the project scope. As intrusion detection system function is for detecting any intrusion in the network, so the next step is to identify the famous attack in network campus environment. When the famous attacks type is identified, it will acquire us to find ways on how to detect that attacks if it is exist in our network. In this case, we need to know its signatures and then, we can detect it by analyzing the packet sent and packet received.

Next step is to setup the intrusion detection system in the campus network environment. I have downloaded one of open source software from the Internet for my IDS. The open source software that I use is Snort where it has some rules to detect an intrusion in the network. This rules is created based on the attacks signature and if the signature in the packet is suit or match with the specified signature, then it will display an alert message that tell us what and where it is happen. In order to setup the IDS, I run the IDS on my PC which is connected to the others who are in the same area with me. Then, IDS will analyze the packet that is

sent through the network and it will store it in the log files. If there were an alert for the packet sent, then it will be stored in the alert.ids files in bin folder. Here is the view of some PCs that existed in this network:

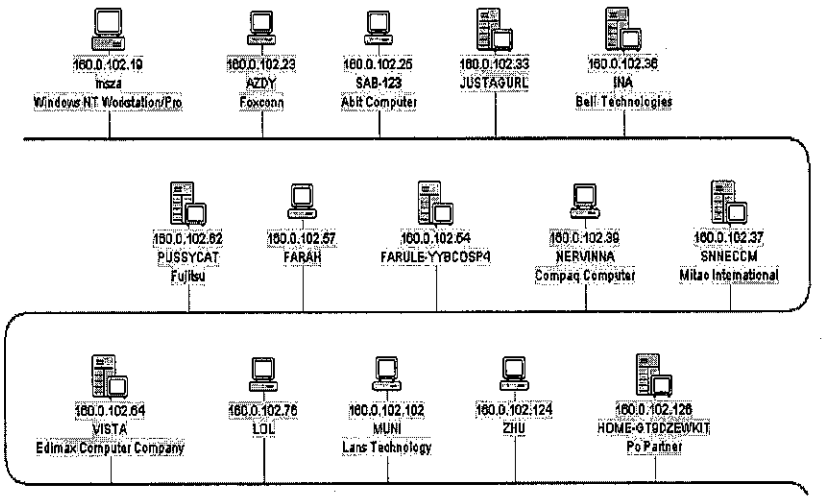


Figure 1: Network View

As I apply Snort for my IDS, so there were some configuration that need to be done according to the network environment where we need to specify the internal and external network addresses.

Fourth step in this project is to simulate the attacks in the network. In this step, I need to simulate the attacks in the network and understand on how it happens and what it is all about. Then, the last step for this project is to measure the effectiveness of the IDS. In order to measure it effectiveness, I will try to make a simple application that can send attack randomly and then IDS will detect it according to the signature pattern in the packet. The efficiency of the IDS needs to be measure because IDS may send false positive alert to the user or to the network administrator. So, this simple application is used to ensure that the attack that is sent into the network is detected by the IDS. Then, the attack that is sent into the network will be stored in the log files for further references and it also will be used to compare with the Snort log files.

After IDS is run, I have some results that need to be analyzed. This result contains the information from where and to where the data is transferred and it also shows on

which protocol that person used to transfer the data. The result gain needs to be analyzed carefully because it may send false positive alert to the administrator.

In order to complete this project, it may require some tools that will be used in order to setup network with IDS and it also will help me in understanding the concept besides teach me something new about IDS. Some tools that I used in completing this project are:

- Personal Computer
- Open Source Software - Snort
- Network Tools – such as network card, network cable, router or switches

3.1 Types of Network Attack

Nowadays, there were various type of attacks occurs in the network environment. Attacks generally can be categorized in two areas which are passive and active. A passive attack usually aimed at gaining access to penetrate the system without compromising IT resources while an active attack results in an unauthorized state change of IT resources. Most of attacks is coming from own enterprise's employees or their business partners or customers which are internal or inside the network. If the attacks is coming from outside or external which frequently via Internet, firewall can help us to detect it because most of firewall used is designed to detect any unauthorized activities from network outside.

Most of attack is not a single action but it is a series of individual events that is developed in a coordinated manner. Here are some types of attacks that occur in the network:

- Password cracking

  It is an action that has been done to crack our password for any application we used without our permission. The password can be cracked by using the password cracker. A password cracker is an application program that is used to identify an unknown or forgotten password to a computer or network resources. It can also be used to help a human cracker obtain unauthorized

access to resources. Password crackers use two primary methods to identify correct passwords which are brute-force and dictionary searches. When a password cracker uses brute-force, it runs through combinations of characters within a predetermined length until it finds the combination accepted by the computer system. When conducting a dictionary search, a password cracker searches each word in the dictionary for the correct password. Password dictionaries exist for a variety of topics and combinations of topics, including politics, movies, and music groups. Some password cracker programs search for hybrids of dictionary entries and numbers. For example, a password cracker may search for ants01; ants02; ants03, etc. This can be helpful where users have been advised to include a number in their password. A password cracker may also be able to identify encrypted passwords. After retrieving the password from the computer's memory, the program may be able to decrypt it. Or, by using the same algorithm as the system program, the password cracker creates an encrypted version of the password that matches the original.

- Malicious code such as virus, worms

The two most common types of malicious code attacks are the virus and the worm. A virus is a program used to infect a computer. It is usually buried inside another program known as a Trojan or distributed as a stand-alone executable. Not all viruses are malicious in fact it is very few that cause extensive damage to systems. Most viruses are simply practical jokes, designed to make it appear, or scare recipients into thinking, that something is wrong with Windows. Unfortunately, the viruses that are destructive are often extremely destructive. A well-designed virus can disable an entire network in a matter of minutes. Worms are often confused with viruses, but they are very different types of code. A worm is self-replicating code that spreads itself from system to system. A traditional virus requires manual intervention to propagate itself, by copying it unknowingly to a floppy, unwittingly embedding it in an attachment, or some other method. Worms do not require assistance to spread because a worm can automatically e-mail itself to other users, copy itself through the network, or even scan other hosts

for vulnerabilities and then attack those hosts. A worm resides in active memory where the program is executed, it will do what it is going to do, and propagates itself. A virus typically overwrites, or attaches itself to, system files. The distinction is often difficult to follow. It is not uncommon for a virus to be paired with a worm prior to launch. The virus does its job, and the worm transports the virus to the next group of victims. Worms have become much more dangerous with the advent of application integration. Many worms take advantage of code that allows programs to automatically execute code to automate common office tasks. E-mail applications are often especially vulnerable to worms. Sometimes worms are sent as attachments that execute when a user attempts to open them, but more often the malicious code can be executed simply by previewing the message, without even reading the message.

- Man in the middle attacks

Men in the middle attacks are much more onerous. Here, the attacker intercepts traffic heading between two devices on the network. The attacker can either monitor information or alter the data as it passes through the network. Typically a man in the middle attack works like this: An attacker sits on the network and watches traffic. When another user on the network sends an ARP request to a network device, the attacker sends a response saying the compromised machine is the requested device. Even if the actual device responds, the second response will override the first. The user now sends all data destined for the original device to the compromised machine. It is possible for an attacker to use this method to intercept enough data to effectively monitor and log all network traffic and gain important information such as usernames and passwords. Users may never know that the traffic is being intercepted, because each packet will eventually be forwarded onto its intended destination.

- Spoofing

  An IP spoofing attack is one in which the source IP address of a packet is forged. There are generally two types of spoofing attacks: IP spoofing used in DoS attacks and man in the middle attacks. IP spoofing-based DoS attacks are relatively straightforward. An attacker sends a packet to the target host with a forged IP address (SYN)—often an IP address in the RFC 1918 address space, though it does not have to be. The targeted host sends an acknowledgement (ACK) and waits for a response. The response never comes, and these unanswered queries remain in the buffer of the targeted device. If enough spoofed queries are sent, the buffer will overflow and the network device will become unstable and crash.

- Brute force attack

  It is a type of password attack that does not attempt to decrypt any information but simply continue to try different passwords. For example a brute-force attack may have a dictionary of all words and/or a listing of commonly used passwords. To gain access to the account using a brute-force attack the program would try all the available words it has to gain access to the account. Another type of brute-force attack is a program that runs through all letters and/or letters and numbers until it gets a match. Although a brute-force attack may be able to gain access to an account eventually, these types of attacks can take several hours, days, months, and even years to run. The amount of time it takes to complete these attacks is dependent on how complicated the password is. So it is advice to all users to change their password frequently in order to avoid this attack.

- Scanning ports and services

  It includes ICMP scanning, UDP, and TCP Stealth Scanning. Prior to sniffing a network an attacker has to gain access. Attackers gain access by scanning devices on the network for vulnerabilities, then exploiting them. Port scanning can either be targeted or random. An attacker interested in a particular network will attempt to track down information about that network

and scan for vulnerabilities. Alternatively, attackers will put large net blocks into a port scanner and let it run for days, trying to find any machine that is available and able to be exploited. This highlights the difference between an attacker and script kiddies.

3.2 IDS Architecture

As to setup my IDS in the network, I have design its architecture based on the network. I have installed Snort in workstation no.5 (WS5) in the network, then the IDS will examine the packet that is sent or received trough the network.

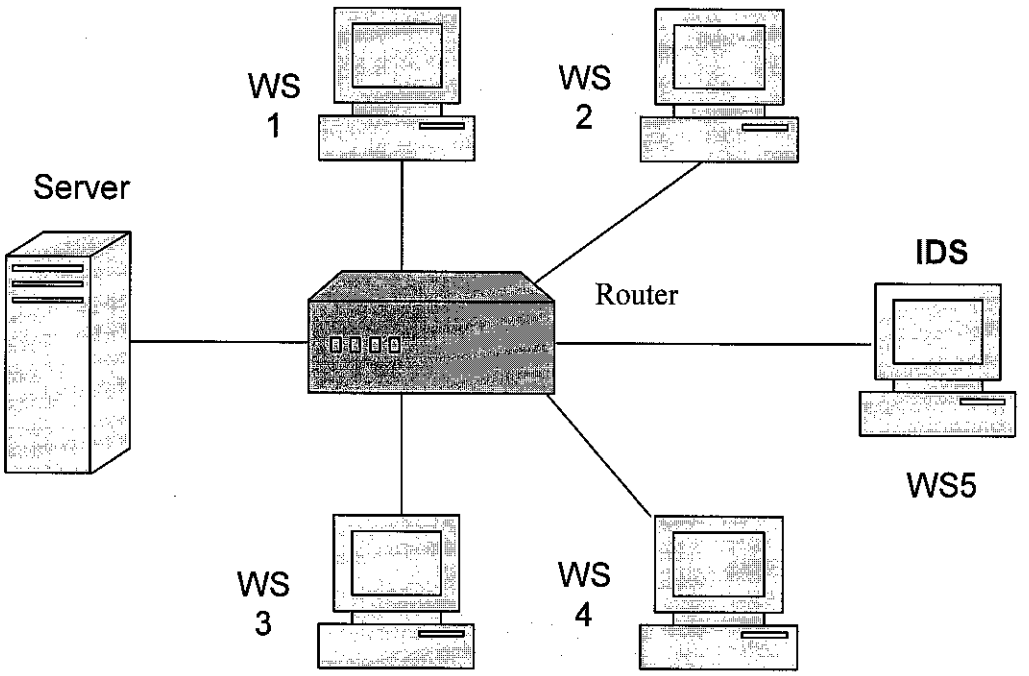


Figure 2: IDS Architecture

CHAPTER 4
RESULTS AND DISCUSSION

Snort can be run in various modes from simply dumping sniffed traffic to the screen where it is able to compare the network traffic with a pre-configured set of signatures known as rules that are house in one or more files. Snort is typically run on command line whether it is run on Windows or Unix host. The most common practical command line options that has been used is –c snort.conf where it allows user place snort in NIDS mode by informing it of the configuration file to be used. The writer must customize this file for her site where this file is provided in the Snort download directory.

Snort allows action to be assigned to each rule, indicating what to do when the rule is triggered. Here is an example of a Snort file entry:

```
[**] NMAP TCP ping [**]
03/12 - 13:33.51:880120 1.2.3.4:1029 -> 192.168.5.5:80
TCP TTL:46 TOS:0x0 ID:19678
******A* Seq : 0xE4F00003 Ack: 0x0 Win: 0xC00
```

There is identifying message associated with the alert that the user can assign when the rule is created or inform the analyst of the perceiving problem. The message for the preceding alert is "NMAP TCP ping". Next is the date and timestamp followed by IP address and port number. The arrow is the direction of traffic where it shows the source is at left and the destination is on the right where the IP address and port number is already specified after the arrow. The third line indicates that the traffic is TCP, which has an arriving time-to-live (46), type of service value (0) and IP identification number (19678). In the final line, it list the TCP flag set. 'A' signifies that the acknowledgement flag is set followed by hexadecimal representation of TCP sequence number, the acknowledgement number and the TCP window size. All of these fields can provide more details about the packet that trigger the alert.

This alert appears because there is a rule examines TCP segments with an acknowledgement flag set but an accompanying acknowledgement value of 0. Most of the time, when this is observed, it is telltale sign of nmap attempting to discover live host. If the acknowledgement is allowed to reach the destination host, the host should respond to the unsolicited acknowledgement with a reset, regardless of whether the port is listening or not. That is why the message accompanying the alert is "NMAP TCP ping".

Snort supports both header and payload inspection methods, which allow you to fully specify in a single rule. There are many but the most important is the capability to inspect and alter signatures. If the analyst can examine the signatures and the packet that caused alert, there is a better chance to make a more accurate assessment. Additionally, signatures that allow an analyst to look at any field from different perspectives potentially improve the quality of IDS. In other words, it only allow writer to create rules that inspect packets for a given IP, port or protocol however it lacks the range to examine payloads or header fields on more granular level.

Benefit that we can gain by using Snort is it comes in a large set of rules. But then it is not recommended to use all the rules on installation because the more active rules used, the network traffic inspection becomes slower. So, I need to decide which rules are appropriate for a campus network depend of the types of attack that occur.

The Snort format for defining packet signatures is now a de-facto standard for publishing signatures. A rules file in Snort is constructed by collecting together a set of rules that are applicable for a particular site. Simply collecting together the thousands of Snort rules that have been defined is not a good strategy. Many of the rules will generate false alert, reduce the network performance where the network bandwidth will become slower and it also will reduce the value of the monitoring system. In addition, the Snort application is designed to operate in the position of the firewall, examining all the traffic on and off site in order to prevent unauthorized access from outside. In this project, I assume that there is a firewall in place and my job is to identify where the local hosts become compromised.

Most of the rules files which is located in rules folder is triggering the packet from external network rather than internal network. Both of internal and external network have been define in the configuration files where the external network means the other packet that is sent by someone from other network that is not equal to internal network.

CHAPTER 5
CONCLUSION AND RECOMMENDATION

Intrusion detection systems have generated a great deal of attention in the security world. They present a vision of an ever-vigilant system sentinel, equipped with the capability to assimilate quantities of information generated by complex system. Intrusion detection systems allow users to optimize the penetrate-and-patch process by sharing penetration knowledge between sites. Many users acknowledge that penetrate and patch is not an optimal solution to system security problems. However, it is the only available option for the user to perform secure design. In addition, the event monitoring and attack recognition capabilities of intrusion detection system enhance the security of a system in other ways. First, these capabilities have a significant deterrent effect on attackers and it also can deal with insider threat. Second, automated responses can disrupt some attacks at the outset making the subsequent attacks more difficult to stage. Third, monitoring the operation of the rest of the security infrastructure allows system security managers to see when security protections are not functioning properly and to rectify the situation before an attacker exploits. And finally, the information gained by monitoring the system can sometimes make it easier to manage the system in other ways, which will result in greater system reliability.

REFERENCES

[1] Broucek,V. & Turner, P (2003) "Intrusion Detection: Forensic Computing Insights Arising From A Case Study on SNORT. In U.E Gattiker(Ed.), EICAR Conference Best Paper

[2] Pete Herzog, 2001, "Open Source Security Testing Methodology Manual"

[3] Vic Lomet, "Using Network Troubleshooting Tools to Help with Network Security: Solving Real-World Security Problems with Sniffer Portable and Sniffer Distributed"

[4] Robert Graham, 14 Sept 2000 http://www.robertgraham.com/pubs/sniffing-faq.html

[5] Stephen Northcut, Judy Novak, Network Intrusion Detection, 3rd Edition, New Riders

[6] Dr. Michael E.Whitman, Herbert J.Mattord, Principles of Information Security, Thomson Course Technology

[7] James M.Kretchman, Open Source Network Administration, Prentice Hall

[8] http://www.snort.org

```
# Configure your server lists.  This allows snort to only look for
#attacks to systems that have a service up.  Why look for HTTP
#attacks if you are not running a web server?  This allows quick
#filtering based on IP addresses
# These configurations MUST follow the same configuration scheme as
#defined above for $HOME_NET.

# List of DNS servers on your network
var DNS_SERVERS $HOME_NET

# List of SMTP servers on your network
var SMTP_SERVERS $HOME_NET

# List of web servers on your network
var HTTP_SERVERS $HOME_NET

# List of sql servers on your network
var SQL_SERVERS $HOME_NET

# List of telnet servers on your network
var TELNET_SERVERS $HOME_NET

# List of snmp servers on your network
var SNMP_SERVERS $HOME_NET

# Configure your service ports.  This allows snort to look for
#attacks destined to a specific application only on the ports that
#application runs on.  For example, if you run a web server on port
#8081, set your HTTP_PORTS variable like this:
#
# var HTTP_PORTS 8081
#
# Port lists must either be continuous [eg 80:8080], or a single
#port [eg 80].
# We will adding support for a real list of ports in the future.

# Ports you run web servers on
#
# Please note:  [80,8080] does not work.
# If you wish to define multiple HTTP ports,
#
# var HTTP_PORTS 80
# include somefile.rules
# var HTTP_PORTS 8080
# include somefile.rules
var HTTP_PORTS 80

# Ports you want to look for SHELLCODE on.
var SHELLCODE_PORTS !80

# Ports you do oracle attacks on
var ORACLE_PORTS 1521

# other variables
#
# AIM servers.  AOL has a habit of adding new AIM servers, so
#instead of modifying the signatures when they do, we add them to
#this list of servers.
```

```
var                                                          AIM_SERVERS
[64.12.24.0/24,64.12.25.0/24,64.12.26.14/24,64.12.28.0/24,64.12.29.0
/24,64.12.161.0/24,64.12.163.0/24,205.188.5.0/24,205.188.9.0/24]

# Path to your rules files (this can be a relative path)
# Note for Windows users:  You are advised to make this an absolute
#path,
# such as:  c:\snort\rules
var RULE_PATH c:\snort\rules

# Configure the snort decoder
# ============================
#
# Snort's decoder will alert on lots of things such as header
# truncation or options of unusual length or infrequently used tcp
#options
#
#
# Stop generic decode events:
#
# config disable_decode_alerts
#
# Stop Alerts on experimental TCP options
#
# config disable_tcpopt_experimental_alerts
#
# Stop Alerts on obsolete TCP options
#
# config disable_tcpopt_obsolete_alerts
#
# Stop Alerts on T/TCP alerts
#
# In snort 2.0.1 and above, this only alerts when a TCP option is
#detected that shows T/TCP being actively used on the network.  If
#this is normal behavior for your network, disable the next option.
#
# config disable_tcpopt_ttcp_alerts
#
# Stop Alerts on all other TCPOption type events:
#
# config disable_tcpopt_alerts
#
# Stop Alerts on invalid ip options
#
# config disable_ipopt_alerts

# Configure the detection engine
# ===============================
#
# Use a different pattern matcher in case you have a machine with
#very limited resources:
#
# config detection: search-method lowmem

##################################################
# Step #2: Configure preprocessors
#
# General configuration for preprocessors is of
# the form
# preprocessor <name_of_processor>: <configuration_options>
```

```
# Configure Flow tracking module
# ------------------------------
#
# The Flow tracking module is meant to start unifying the state
#keeping mechanisms of snort into a single place. Right now, only a
#portscan detector is implemented but in the long term, many of the
#stateful subsystems of snort will be migrated over to becoming flow
#plugins. This must be enabled for flow-portscan to work correctly.
#
# See README.flow for additional information
#
preprocessor flow: stats_interval 0 hash 2

# frag2: IP defragmentation support
# ---------------------------------
# This preprocessor performs IP defragmentation.  This plugin will
#also detect people launching fragmentation attacks (usually DoS)
#against hosts.  No arguments loads the default configuration of the
#preprocessor, which is a 60 second timeout and a 4MB fragment
#buffer.

# The following (comma delimited) options are available for frag2
#  timeout [seconds] - sets the number of [seconds] that an
#unfinished fragment will be kept around waiting for completion,
#if this time expires the fragment will be flushed
#    memcap [bytes] - limit frag2 memory usage to [number] bytes
#                     (default:  4194304)
#
#    min_ttl [number] - minimum ttl to accept
#
#      ttl_limit [number] - difference of ttl to accept without
#alerting will cause false positves with router flap
#
# Frag2 uses Generator ID 113 and uses the following SIDS
# for that GID:
#  SID     Event description
# -----   -------------------
#   1       Oversized fragment (reassembled frag > 64k bytes)
#   2       Teardrop-type attack


# stream4: stateful inspection/stream reassembly for Snort
#-----------------------------------------------------------------
---
# Use in concert with the -z [all|est] command line switch to defeat
#stick/snot against TCP rules.  Also performs full TCP stream
#reassembly, stateful inspection of TCP streams, etc.  Can
#statefully detect various portscan types, fingerprinting, ECN, etc.

# stateful inspection directive
# no arguments loads the defaults (timeout 30, memcap 8388608)
# options (options are comma delimited):
# detect_scans - stream4 will detect stealth portscans and generate
#alerts when it sees them when this option is set
#detect_state_problems - detect TCP state problems, this tends to be
#very noisy because there are a lot of crappy ip stack
#implementations out there
#
```

```
#disable_evasion_alerts - turn off the possibly noisy mitigation of
#overlapping sequences.
#
#
#   min_ttl [number]       - set a minium ttl that snort will accept
#to stream reassembly
#
#    ttl_limit [number]      - differential of the initial ttl on a
#session versus the normal that someone may be playing games.
# Routing flap may cause lots of false positives.
#
#     keepstats [machine|binary] - keep session statistics, add
#"machine" to get them in a flat format for machine reading, add
#"binary" to get them in a unified binary output
#                          format
#noinspect - turn off stateful inspection only timeout [number] -
#set the session timeout counter to [number] seconds,
#default is 30 seconds
#   memcap [number] - limit stream4 memory usage to [number] bytes
#    log_flushed_streams - if an event is detected on a stream this
#option will cause all packets that are stored in the stream4
#packet buffers to be flushed to disk.  This only works when logging
#in pcap mode!
#
# Stream4 uses Generator ID 111 and uses the following SIDS
# for that GID:
#  SID     Event description
# -----   -------------------
#   1       Stealth activity
#   2       Evasive RST packet
#   3       Evasive TCP packet retransmission
#   4       TCP Window violation
#   5       Data on SYN packet
#   6       Stealth scan: full XMAS
#   7       Stealth scan: SYN-ACK-PSH-URG
#   8       Stealth scan: FIN scan
#   9       Stealth scan: NULL scan
#   10      Stealth scan: NMAP XMAS scan
#   11      Stealth scan: Vecna scan
#   12      Stealth scan: NMAP fingerprint scan stateful detect
#   13      Stealth scan: SYN-FIN scan
#   14      TCP forward overlap

preprocessor stream4: detect_scans, detect_state_problems,ttl_limit
150

# tcp stream reassembly directive
# no arguments loads the default configuration
#   Only reassemble the client,
#   Only reassemble the default list of ports (See below),
#   Give alerts for "bad" streams
#
# Available options (comma delimited):
#     clientonly - reassemble traffic for the client side of a
#connection only
#     serveronly - reassemble traffic for the server side of a
#connection only
#   both - reassemble both sides of a session
#    noalerts - turn off alerts from the stream reassembly stage of
#stream4
```

```
#   ports [list] - use the space separated list of ports in [list],
#"all"
#                  will turn on reassembly for all ports, "default"
#will turn
#                  on reassembly for ports 21, 23, 25, 53, 80, 143,
#110, 111
#                 and 513

preprocessor stream4_reassemble:both

# http_inspect: normalize and detect HTTP traffic and protocol
#anomalies
#
# lots of options available here. See doc/README.http_inspect.
# unicode.map should be wherever your snort.conf lives, or given
# a full path to where snort can find it.
preprocessor http_inspect: global iis_unicode_map unicode.map 1252

preprocessor http_inspect_server: server default \
    profile all ports { 80 8080 8180 } oversize_dir_length 500

#
#  Example unqiue server configuration
#
#preprocessor http_inspect_server: server 1.1.1.1 \
#    ports { 80 3128 8080 } \
#    flow_depth 0 \
#    ascii no \
#    double_decode yes \
#    non_rfc_char { 0x00 } \
#    chunk_length 500000 \
#    non_strict \
#    oversize_dir_length 300 \
#    no_alerts


# rpc_decode: normalize RPC traffic
# ----------------------------------
# RPC may be sent in alternate encodings besides the usual 4-byte
#encoding
# that is used by default. This plugin takes the port numbers that
#RPC
# services are running on as arguments - it is assumed that the
#given ports
# are actually running this type of service. If not, change the
#ports or turn
# it off.
# The RPC decode preprocessor uses generator ID 106
#
# arguments: space separated list
# alert_fragments - alert on any rpc fragmented TCP data
# no_alert_multiple_requests - don't alert when >1 rpc query is in a
#packet
# no_alert_large_fragments - don't alert when the fragmented
#                            sizes exceed the current packet size
# no_alert_incomplete - don't alert when a single segment
#                       exceeds the current packet size

preprocessor rpc_decode: 111 32771

# bo: Back Orifice detector
```

```
# -------------------------
# Detects Back Orifice traffic on the network.  Takes no arguments
#in 2.0.
#
# The Back Orifice detector uses Generator ID 105 and uses the
# following SIDS for that GID:
#  SID     Event description
# -----   -------------------
#   1       Back Orifice traffic detected

preprocessor bo

# telnet_decode: Telnet negotiation string normalizer
# ---------------------------------------------------
# This preprocessor "normalizes" telnet negotiation strings from
#telnet and ftp
# traffic.   It works in much the same way as the http_decode
#preprocessor,
# searching for traffic that breaks up the normal data stream of a
#protocol and
# replacing it with a normalized representation of that traffic so
#that the
# "content" pattern matching keyword can work without requiring
#modifications.
# This preprocessor requires no arguments.
# Portscan uses Generator ID 109 and does not generate any SID
#currently.

preprocessor telnet_decode

# Flow-Portscan: detect a variety of portscans
# --------------------------------------
# Note:  The Flow preprocessor (above) must first be enabled for
#Flow-Portscan to
# work.
#
# This module detects portscans based off of flow creation in the
#flow
# preprocessors.  The goal is to catch one->many hosts and one->many
# ports scans.
#
# Flow-Portscan has numerous options available, please read
# README.flow-portscan for help configuring this option.

# Flow-Portscan uses Generator ID 121 and uses the following SIDS
#for that GID:
#  SID     Event description
# -----   -------------------
#   1       flow-portscan: Fixed Scale Scanner Limit Exceeded
#   2       flow-portscan: Sliding Scale Scanner Limit Exceeded
#   3       flow-portscan: Fixed Scale Talker Limit Exceeded
#   4   flow-portscan: Sliding Scale Talker Limit Exceeded

# preprocessor flow-portscan: \
#     talker-sliding-scale-factor 0.50 \
#     talker-fixed-threshold 30 \
#     talker-sliding-threshold 30 \
#     talker-sliding-window 20 \
#     talker-fixed-window 30 \
#     scoreboard-rows-talker 30000 \
#     server-watchnet [10.2.0.0/30] \
```

```
#     server-ignore-limit 200 \
#     server-rows 65535 \
#     server-learning-time 14400 \
#     server-scanner-limit 4 \
#     scanner-sliding-window 20 \
#     scanner-sliding-scale-factor 0.50 \
#     scanner-fixed-threshold 15 \
#     scanner-sliding-threshold 40 \
#     scanner-fixed-window 15 \
#     scoreboard-rows-scanner 30000 \
#     src-ignore-net [192.168.1.1/32,192.168.0.0/24] \
#     dst-ignore-net [10.0.0.0/30] \
#     alert-mode once \
#     output-mode msg \
#     tcp-penalties on

# arpspoof
#--------------------------------------
# Experimental ARP detection code from Jeff Nathan, detects ARP
#attacks,
# unicast ARP requests, and specific ARP mapping monitoring.  To
#make use of
# this preprocessor you must specify the IP and hardware address of
#hosts on
# the same layer 2 segment as you.  Specify one host IP MAC combo
#per line.
# Also takes a "-unicast" option to turn on unicast ARP request
#detection.
# Arpspoof uses Generator ID 112 and uses the following SIDS for
#that GID:

#  SID     Event description
# -----   -------------------
#   1       Unicast ARP request
#   2       Etherframe ARP mismatch (src)
#   3       Etherframe ARP mismatch (dst)
#   4       ARP cache overwrite attack

#preprocessor arpspoof
#preprocessor arpspoof_detect_host: 192.168.40.1 f0:0f:00:f0:0f:00
preprocessor arpspoof_detect_host: 192.168.40.1 f0:0f:00:f0:0f:00


# Performance Statistics
# ----------------------
# Documentation for this is provided in the Snort Manual.  You
#should read it.
# It is included in the release distribution as doc/snort_manual.pdf
#
# preprocessor perfmonitor: time 300 file /var/snort/snort.stats
#pktcnt 10000

##################################################################
# Step #3: Configure output plugins
#
# Uncomment and configure the output plugins you decide to use.
#General
# configuration for output plugins is of the form:
#
# output <name_of_plugin>: <configuration_options>
#
```

```
# alert_syslog: log alerts to syslog
# ----------------------------------
# Use one or more syslog facilities as arguments.  Win32 can also
#optionally
# specify a particular hostname/port.  Under Win32, the default
#hostname is
# '127.0.0.1', and the default port is 514.
#
# [Unix flavours should use this format...]
# output alert_syslog: LOG_AUTH LOG_ALERT
#
# [Win32 can use any of these formats...]
# output alert_syslog: LOG_AUTH LOG_ALERT
# output alert_syslog: host=hostname, LOG_AUTH LOG_ALERT
# output alert_syslog: host=hostname:port, LOG_AUTH LOG_ALERT

# log_tcpdump: log packets in binary tcpdump format
# -------------------------------------------------
# The only argument is the output file name.
#
 output log_tcpdump: snort.log

# database: log to a variety of databases
# ---------------------------------------
#  See the README.database file for more information about
#configuring
# and using this plugin.
#
# output database: log, mysql, user=root password=test dbname=db
#host=localhost
# output database: alert, postgresql, user=snort dbname=snort
# output database: log, odbc, user=snort dbname=snort
# output database: log, mssql, dbname=snort user=snort password=test
#  output database: log, oracle, dbname=snort user=snort
#password=test

# unified: Snort unified binary format alerting and logging
# -------------------------------------------------------------
# The unified output plugin provides two new formats for logging and
#generating
# alerts from Snort, the "unified" format.  The unified format is a
#straight
# binary format for logging data out of Snort that is designed to be
#fast and
# efficient.  Used with barnyard (the new alert/log processor), most
#of the
# overhead for logging and alerting to various slow storage
#mechanisms such as
# databases or the network can now be avoided.
#
# Check out the spo_unified.h file for the data formats.
#
# Two arguments are supported.
#      filename - base filename to write to (current time_t is
#appended)
#    limit    - maximum size of spool file in MB (default: 128)
#
# output alert_unified: filename snort.alert, limit 128
# output log_unified: filename snort.log, limit 128
```

```
# You can optionally define new rule types and associate one or more
#output
# plugins specifically to that type.
#
# This example will create a type that will log to just tcpdump.
# ruletype suspicious
# {
#   type log
#   output log_tcpdump: suspicious.log
# }
#
# EXAMPLE RULE FOR SUSPICIOUS RULETYPE:
# suspicious tcp $HOME_NET any -> $HOME_NET 6667 (msg:"Internal IRC
#Server";)
#
# This example will create a rule type that will log to syslog and a
#mysql
# database:
# ruletype redalert
# {
#   type alert
#   output alert_syslog: LOG_AUTH LOG_ALERT
#       output database: log, mysql, user=snort dbname=snort
#host=localhost
# }
#
# EXAMPLE RULE FOR REDALERT RULETYPE:
# redalert tcp $HOME_NET any -> $EXTERNAL_NET 31337 \
#   (msg:"Someone is being LEET"; flags:A+;)

#
# Include classification & priority settings
# Note for Windows users:  You are advised to make this an absolute
#path,
# such as:  c:\snort\etc\classification.config
#
#var RULE_PATH c:\snort\etc\

var RULE_PATH c:\snort\etc
include $RULE_PATH\classification.config

#
# Include reference systems
# Note for Windows users:  You are advised to make this an absolute
#path,
# such as:  c:\snort\etc\reference.config
#
# var RULE_PATH c:\snort\etc\

var RULE_PATH c:\snort\etc\

include $RULE_PATH\reference.config

##################################################################
# Step #4: Customize your rule set
#
# Up to date snort rules are available at http://www.snort.org
#
# The snort web site has documentation about how to write your own
#custom snort
# rules.
```

```
#
# The rules included with this distribution generate alerts based on
#on
# suspicious activity. Depending on your network environment, your
#security
# policies, and what you consider to be suspicious, some of these
#rules may
# either generate false positives ore may be detecting activity you
#consider to
# be acceptable; therefore, you are encouraged to comment out rules
#that are
# not applicable in your environment.
#
#  The following individuals contributed many of rules in this
#distribution.
#
#=========================================
# Include all relevant rulesets here
#
# The following rulesets are disabled by default:
#
#   web-attacks, backdoor, shellcode, policy, porn, info, icmp-info,
#virus,
#   chat, multimedia, and p2p
#
# These rules are either site policy specific or require tuning in
#order to not
# generate false positive alerts in most environments.
#
# Please read the specific include file for more information and
# README.alert_order for how rule ordering affects how alerts are
#triggered.
#=========================================
var RULE_PATH c:\snort\rules

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

# Include any thresholding or suppression commands. See
#threshold.conf in the
# <snort src>/etc directory for details. Commands don't necessarily
#need to be
# contained in this conf, but a separate conf makes it easier to
#maintain them.
# Note for Windows users:  You are advised to make this an absolute
#path,
# such as:  c:\snort\etc\threshold.conf
# Uncomment if needed.
# include threshold.conf
```

Appendix 2 : Snort Command Utilities

USAGE: snort [-options] <filter options>

snort /SERVICE /INSTALL [-options] <filter options>

snort /SERVICE /UNINSTALL

snort /SERVICE /SHOW

Options:

-A Set alert mode: fast, full, console, or none (alert file alerts only)

-b Log packets in tcpdump format (much faster!)

-c <rules> Use Rules File <rules>

-C Print out payloads with character data only (no hex)

-d Dump the Application Layer

-e Display the second layer header info

-E Log alert messages to NT Eventlog. (Win32 only)

-f Turn off fflush() calls after binary log writes

-F <bpf> Read BPF filters from file <bpf>

-h <hn> Home network = <hn>

-i <if> Listen on interface <if>

-I Add Interface name to alert output

-k <mode> Checksum mode (all,noip,notcp,noudp,noicmp,none)

-l <ld> Log to directory <ld>

-L <file> Log to this tcpdump file

-n <cnt> Exit after receiving <cnt> packets

-N Turn off logging (alerts still work)

-o Change the rule testing order to Pass|Alert|Log

-O Obfuscate the logged IP addresses

-p Disable promiscuous mode sniffing

-P <snap> Set explicit snaplen of packet (default: 1514)

-q Quiet. Don't show banner and status report

-r <tf> Read and process tcpdump file <tf>

-R <id> Include 'id' in snort_intf<id>.pid file name

-s Log alert messages to syslog

-S <n=v> Set rules file variable n equal to value v

-T Test and report on the current Snort configuration

-U Use UTC for timestamps

-v Be verbose

-V Show version number

-W Lists available interfaces. (Win32 only)

-w Dump 802.11 management and control frames

-X Dump the raw packet data starting at the link layer

-y Include year in timestamp in the alert and log files

-z Set assurance mode, match on established sesions (for TCP)

-? Show this information

<Filter Options> are standard BPF options, as seen in TCPDump