Florida State University Libraries

Electronic Theses, Treatises and Dissertations The Graduate School

2004

Ramit-Rule-Based Alert Management Information Tool

John Blackwell

Follow this and additional works at the FSU Digital Library. For more information, please contact lib-ir@fsu.edu

THE FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

RAMIT –

RULE-BASED ALERT MANAGEMENT INFORMATION TOOL

By

JOHN BLACKWELL

A Thesis submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Master of Science

Degree Awarded:
Summer Semester, 2004

Copyright © 2004
John Blackwell
All Rights Reserved

The members of the Committee approve the thesis of John Blackwell defended on May 28, 2004.

_____
Lois Wright Hawkes
Professor Directing Thesis

_____
Michael Burmester
Committee member

_____
Daniel G. Schwartz
Committee member

The Office of Graduate Studies has verified and approved the above named committee members.

For these individuals:

My wife, Lisa Blackwell, for her loving assistance every step of the way.
My aunt, Linda Watkins for putting the learning “bug” in me .
My uncle, Ray Watkins, who watched out for me when I couldn’t watch out for myself.
And most of all, Jesus Christ without whom any of this would make any sense whatsoever.

Thank you all!

ACKNOWLEDGEMENTS

I would like to thank my wife Lisa Blackwell for her abundant patience and guidance in the writing of this paper. In fact, it would have been impossible without her assistance.

Thanks also to Ann Sorrenti and Stephanie Crew for their practical and moral support, enabling me to pursue my research.

And most of all, thanks to Dr. Lois Hawkes for her infinite patience, understanding and guidance in the writing of this paper.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The problems inherent to providing security for network systems are relative to the openness and design of network architecture. Typically network security is achieved through the use of monitoring tools based on pattern recognition or behavioral analysis. One of the tools based on pattern recognition is SNORT. SNORT attempts to protect networks by alerting system administrators when network received packets of information match predetermined signatures contained in the SNORT tool. Unfortunately, by the very nature of this design, SNORT operates at the packet data level and has no concept of the specific properties of the network it is trying to protect.

This thesis provides the design of an *alert management tool* which, upon taking SNORT alert signatures as inputs and using a knowledge base of intruders and local Network Systems, attempts to reduce false-positive and negative alerts sent to the system administrator. The major drawback to SNORT is that many false alerts are sent from the SNORT engine, and must then be sifted through and classified by system administrators. This thesis proposes a tool which should lessen this stress and considerably reduce the workload of having to classify alerts by human beings.

CHAPTER ONE

INTRODUCTION

Since the formation of the Internet, intrusion attempts on Network Systems have increased astronomically. With increased security measures, have come increasingly clever attacks by much more sophisticated attackers. Because of this, network intrusion detection systems (NIDS) have become more and more needed. Today if someone has an Internet presence, a firewall should be mandated, and a network intrusion detection system should be a given as well. There are already a number of “ready to run” software options available which attempt to provide some measure of network security. SNORT is an open source NIDS that works by using signature recognition. Signature recognition matches predetermined rules with packets observed by a promiscuous operating ethernet card. When SNORT finds a packet or datagram, it then logs this data and, based upon set up, may send an alert to the system administrator. SNORT performance suffers because of incorrectly alerting on packets which truly do not represent threats. System administrators after tweaking SNORT to monitor their Network Systems still have hundreds if not thousands of these false alerts to deal with. The topic of this thesis is how to deal with this information overload of 'false negatives', as well as ‘false positives’.

The solution in this paper is to create a framework around SNORT which can centralize real-time alert data giving both intruder and vulnerability information to the person responsible for network security. The proposed tool that accomplishes this is RAMIT, Rule-based Alert Management Information Tool. RAMIT is designed to accept input from multiple SNORT sensors, to create a three-dimensional linked list with network vulnerability data, and to maintain a database of intruders and methods they use to attack systems. It will use a modified, 'forward chaining' artificial intelligence rule-based system to make decisions based on network awareness and hacker behavior. Instead of indiscriminately sending every event along to the system

administrator, creating an information overload, only pertinent results are forwarded. Once an event is processed and found to exploit a vulnerability, an alert will be sent to an HTML GUI interface where a system administrator monitoring the display output will be able to respond swiftly and precisely to the attack.

Chapter 2 provides valuable background regarding the architecture and design of intrusion detection systems (IDS). Different classes and types of IDS are outlined and problems with IDS are discussed.

Chapter 3 examines a particular class of IDS known as network intrusion detection systems or NIDS. NIDS operate somewhat differently from standard IDS in that the observation of packets across the network lines is considered more than network hosts themselves. A very successful NIDS named NFR is covered in depth and general problems and solutions discovered over time are included in this discussion.

Chapter 4 analyzes a specific NIDS called SNORT. It is this NIDS that this paper is most concerned with and it will be the tool that our alert management system uses for inputs. SNORT is an open-source NIDS and is arguably the most successful implementation of a NIDS on the market. It is so successful in fact that SNORT is often used as the main detection system for other commercial operations, as it will be for our implementation.

Chapter 5 includes a discussion of rule-based artificial intelligence. Artificial intelligence is covered somewhat and different classes of rule-based systems including expert systems are mentioned. Since our alert management tool uses rule-based artificial intelligence, this discussion is necessary. Finally, separate classes of rule-based systems are diagrammed and explained. The tool designed in this thesis will use one of these classes called forward chaining which will be slightly modified.

Chapter 6 provides a systematic analysis of SNORT performance as the system currently operates, without any enhancements. Several alerts are followed through and log output is provided for the reader to evaluate.

Chapter 7 includes a complete discussion on the problems with NIDS in general. This discussion also examines the theory that online signature based intrusion detection is often flawed in its set up. The conflict between knowledge versus data is discussed and a concrete example using SNORT, argues the point that modification or enhancement must occur for datagram signature based detection to evolve.

Chapter 8 gives a general overview of the proposed solution and describes RAMIT, which stands for *rule-based alert management information tool*. RAMIT takes information from multiple SNORT sensors, correlates this information with known hacker behavior, then analyzes network vulnerability, and finally makes a recommendation for the alert to be processed.

Chapter 9 lays out the complete design of RAMIT including parameters and methods. Examples are given that show the difference between SNORT and SNORT along with RAMIT monitoring its output. Finally, conclusions are given and continuing research in the area is mentioned.

CHAPTER TWO

INTRUSION DETECTION SYSTEMS

An intrusion in computer networking terms is defined as someone (hacker, cracker) attempting to bypass security protocols and infiltrate a network system. The motivation behind this could be something as sinister as stealing confidential data, misusing e-mail for spam, or any number of things for which a system administrator could be held responsible. As shown by the February 2000 DOS (Denial of Service) attacks [CNN00] against the major Internet service providers, the frequency of security incidents are increasing. Even more alarming is the evidence that such attacks are becoming much more intelligent, subversive, and harmful. It has become certain that anyone responsible for a network with an Internet presence is now a potential target, and intrusion detection systems are quickly becoming a necessity.

An intrusion detection system (IDS) is a system designed to systematically detect host attacks on a network. These systems provide a secondary, passive level of security by providing the administrator with critical information about intrusion attempts.

In actuality, the IDS simply alerts the system administrators upon detection of computer attacks or computer misuse. Often an IDS is used in conjunction with a firewall whose sole aim (through packet analysis) is to control the flow of *datagrams* into and out of a network. *Datagrams* are simply the packet bundles of information that computer systems use to communicate with each other over the network. Typically IDS are not intended to block or actively counter attacks, but some newer systems have an active capacity for dealing with threats. Indeed, a very knowledgeable human being should be watching and making value judgments on the ‘alerts’ that the IDS has presented him or her with. While firewalls can be thought of as a border or security perimeter, IDS functions to detect whether that border has been breached. Under no circumstances does an IDS guarantee security, but with proper policies, authentications, and access control, some measure of security can be attained [INMC01].

2.1 Operation and Function of IDS

Intrusion detection systems serve three essential security functions: they strive to monitor, detect, and respond to unauthorized activity by company insiders and outsider intrusion. Intrusion detection systems can use rule-based policies that match packets or analysis of user behavior with predetermined security events. Typically, IDS send out alerts if they detect one of these security incidents [INMC01].

Some IDS not only send alerts but also respond to the threat of an event. These responses usually take the form of removing a user, disabling user accounts, launching scripts, and dropping packets. These IDS are in a special class of intrusion systems called IPS or Intrusion Prevention Systems. It is estimated that up to 85% of security incidents on networks come from inside the network firewall [INMC01]. The other 15% or more come from outside in the form of denial of service attacks, buffer overflows and the like. Some of the most inventive attacks are known as Trojan Horses. These are destructive programs masquerading as harmless applications such as e-mail attachments promising to rid your computer of viruses but instead introducing its own virus onto your computer [WEBO]. Currently, intrusion detection systems are the only tools that system administrators possess for detecting and responding to intrusions once the attacker has bypassed the firewall and has gained access to the network.

There are four major IDS techniques used to detect intruders: **anomaly detection, misuse (or signature) detection, target monitoring,** and **stealth probes** [INMC01].

2.1.1 Anomaly Detection

Anomaly detection searches out abnormal patterns of behavior, where an IDS establishes what is known as a "baseline of normal usage," and then any deviation from that baseline is marked as a possible security problem. For example, a computer that has been logged into at 3 a.m. from an office that closes at 6 p.m. should certainly cause a behavioral alert warning. Another instance might be an employee working in the web development group who suddenly takes great interest in accounting software. This would also constitute a standard deviation in baseline normal behavior.

2.1.2 Signature Detection

Signature detection uses specific patterns to detect similar intrusion attacks. Network intrusion detection systems use signatures or partial strings that match parts of the network packet itself. Once the strings are matched, notification is sent to the proper authorities and the incident is logged. These intrusion attempts mark signatures already programmed into the signature database that match parts of the network packet itself. Once the IDS matches a string, it responds by sending the System Administrator an alert. For instance, if several 'backdoor attempts' are sent from a specific host and the system administrator responded properly, eventually it would be determined, if the computer was an inside host, that a hacker was attempting to infiltrate the computer.

2.1.3 Target Monitoring

Target monitoring searches for modifications of specific files. Often cryptographic caches are calculated and compared to new caches of specific files at regular intervals. If these fail to match, then the IDS considers an intrusion to have occurred. An excellent example of target or integrity monitoring is the Tripwire product (*www.tripwire.com*). Tripwire monitors the following attributes [PEICHU04]:

- File additions, deletions, or modifications
- Flags on files (hidden read-only)
- Access times
- Write times
- Change times
- File size
- Hash checking

2.1.4 Stealth Probes

Finally, **stealth probes** are designed to detect attackers who carry out their attack over prolonged periods of time [INMC01]. These hackers, or crackers, test system vulnerabilities and try to open ports, and after a lengthy period of time, actually begin their attacks. These attacks are very difficult to detect. Stealth probes use wide area sampling and try to correlate port scans with later attacks from the same IP numbers.

Although many versions of IDS exist, there are some terms and general definitions common to all and which will be referred to later. We will call an **alert** a true alarm, one which identifies that the system in question has been significantly attacked. This means that a hacker has made contact or attempted to make contact with a vulnerability inside the network area. A **false positive** is an alarm generated for conditions that do not exist or are no threat inside our network environment. A **false negative** occurs when the IDS does not alert when an alert worthy condition has happened. **Noise** occurs when an IDS alerts on conditions that are either non-threatening or truly not applicable to the network sites which are being monitored; however, the IDS was correct in diagnosing these. In other words, the IDS has not made a mistake but the alert given was of no value. [RM03, pg. 4]

2.2 Introduction to Intrusion Detection Systems

There are two classes of intrusion detection systems: host based and network based. Each class has distinct advantages and disadvantages. Host based IDS survey data on individual nodes of a network, whereas network based IDS examine those packets which are exchanged between network nodes [INMC01].

2.2.1 Host-based IDS

Host based intrusion detection systems can be broken down in roughly four types [PEICHU04]. These include log monitors, integrity monitors, signature scanners and anomaly detectors. Logfile monitors try to detect intrusions by parsing system event logs. Target monitors, signature scanners, and anomaly detectors have already been discussed.

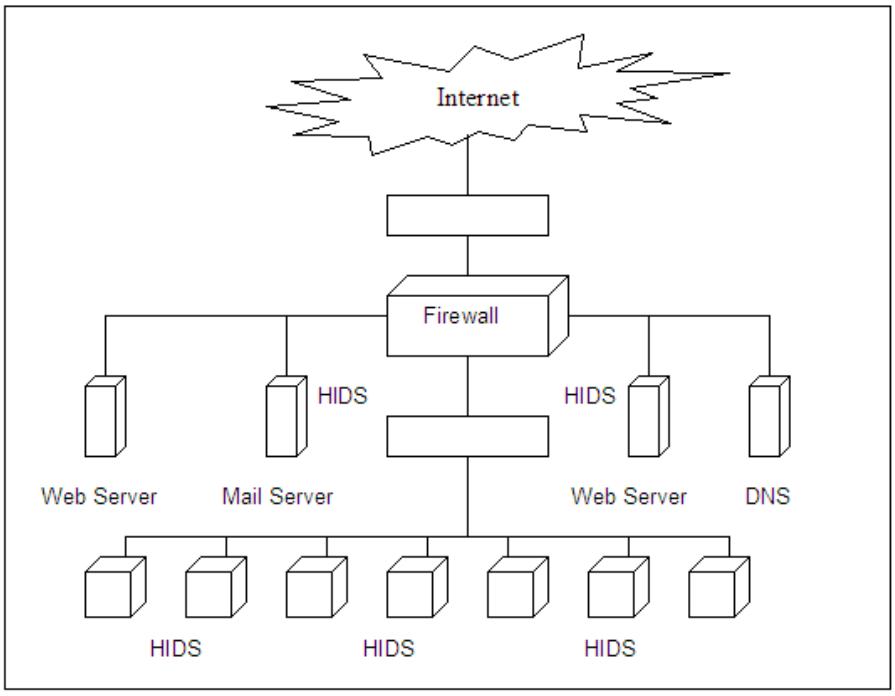

Figure 2.1: Host Intrusion Detection System Schema. Source [CBFP03], 6.

Host based IDS (HIDS), as diagrammed above in Figure 2.1, analyze datagrams which may originate from computers which host services. A good example of this would be a web server. Once the data is captured, the analysis may take place on the host or node or on an analysis machine. One type of HIDS is a program which constantly receives applications or operating system audit logs [INMC01]. These programs 'live' inside a trusted domain within the network and have been proven to be excellent detection devices, being close to network authenticated users. They are fast to respond and usually accurate in detecting unauthorized activity because they are right at the source.

The disadvantage to host based systems is the cumbersome administration and logging attempts made by these programs [INMC01]. If there are several thousand nodes on a large network with each node having its own HIDS, the likelihood of effective and efficient system administration for each of these programs would be impractical and nearly impossible. Also, if an intruder could disable the program or the data collection by the program on any of those given nodes, then the IDS would be completely ineffective in that circumstance.

The alternative to a HIDS is a network based IDS (NIDS), which monitors data traffic throughout the network as opposed to residing on a single machine. The advantages and disadvantages of NIDS will be discussed more comprehensively in Chapter 3.

CHAPTER THREE

NETWORK-BASED INTRUSION DETECTION SYSTEMS

The second class of intrusion detection systems is comprised of network based IDS (NIDS). Network based detection analyzes datagrams traveling over the entire network or switched sections of the entire network. Network packets are analyzed, sometimes compared with pre-stated rules, and then checked for veracity of their nature, whether harmful or benign [INMC01]. The primary technique employed by NIDS is packet sniffing. This approach pulls data from inside certain protocol packets and matches this data with predetermined accepted packet structure.

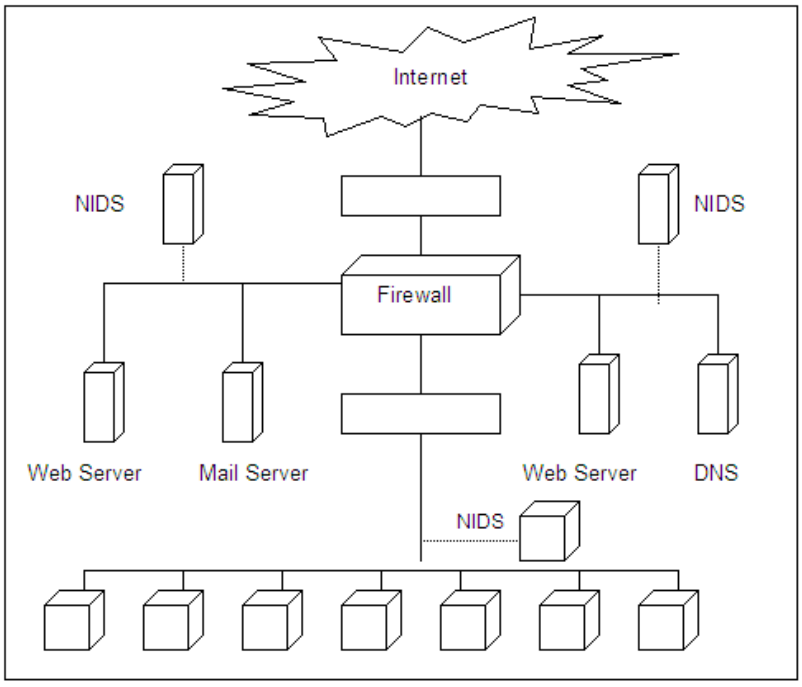

Figure 3.1: Network Intrusion Detection System Schema. Source [CBFP03], 7.

Network based systems are more effective than their host based counterparts at detecting unauthorized outsider access and denial of service or bandwidth theft. However, NIDS do have certain drawbacks, such as a difficulty in analyzing certain packets such as encrypted packet payloads, Unicode formatted packets, high-speed networks, highly switched networks, or any other packet formatted in a way which would make it impossible for a program to match predetermined signatures [INMC01].

3.1 Categories of NIDS

Usually network IDS are broken down into anomaly-based and signature-based categories. Today, the state-of-the-art of NIDS is a combination of both. These are called hybrids. Signature matching NIDS use a database of known attack signatures and attempt to match a hacking attempt exploit to a previously defined rule.

Signature matching NIDS can either be stateful or stateless. Stateless systems deal with individual packets, but stateful systems monitor an entire session or state of a connection. The benefit of stateful systems are that attacks split across multiple datagrams can still be detected using signatures. The following is a simple example of why stateful analysis is so important:

FTP Login

1. User sends “USER jblackwell” to an FTP server, where *jblackwell* is the username.
2. The server responds with “331 Send password”.
3. User sends “PASS token” to the FTP server, where token is the password.
4. FTP server confirms password is correct, and responds with “230 User logged in”. [FRED1569]

The IDS sensor detects the USER command and stores the tentative username for this FTP session [FRED1569]. Once the successful authentication has been detected, the IDS sensor knows that the session it is monitoring is that of username *jblackwell*.

This information is useful to us in two main ways [FRED1569]. First, if an attack occurs during this session, the IDS can report that username *jblackwell* was used for this session. This information may be extremely helpful when investigating an incident [FRED1569]. Second, imagine that a more interesting username than jblackwell was used; perhaps “root” or

"administrator" was used. By statefully analyzing all the authentication-related requests and responses, the IDS sensor can detect attempts to use such accounts, as well as recording whether each was successful or not [FRED1569].

3.1.1 NIDS Examples

The following are taken from a survey of current NIDS and demonstrate what is important to the public as far as system capabilities are concerned. [TIMBER]

- Passively examines network traffic, identifying attacks, probes, and other anomalous events in real-time
- Monitors activity at the server and provides complete auditing and reporting functionality
- Silently monitoring communications between computer and network
- Complete intrusion detection suite that integrates network and host-based intrusion detection vulnerability assessment and audit policy management into a single easy to use package
- Enterprise scale, real-time, intrusion detection system designed to detect report and terminate unauthorized activity
- Detect, report and terminate unauthorized activity
- Ability to detect network reconnaissance stealth port scanning over many months, warning against even the most determined attacks
- Watches live network packets and looks for signs of computer crime, network attacks, network misuse and other anomalies

One of the leading NIDS available today is NFR, a comprehensive package developed by NFR Security which provides all of the features discussed above, and which makes an excellent model for discussion [NFR2003].

3.2 NFR

NFR is becoming a leader in network security for good reason. NFR has a detection engine capable of using advanced signature prospects, protocol anomaly, state tracking and data

context analysis. NFR is built to integrate with most firewalls. Because of this capability, security administrators can set up the NIDS to respond to events with a TCP reset, custom code execution or firewall commands. Also, the custom interface offers alert tuning, which can be used to tweak sensors for reducing false positives. NFR also contains its own signature language called N-Code, and it offers signature libraries for customizing NID systems [NFR2003].

NFR uses advanced signature and stateful protocol analysis for all ethernet based, network based protocol detection types. The concept behind stateful protocol analysis is very simple. When performing protocol analysis on TCP and UDP and other payloads, protocols such as FTP, HTTP, DNS, are examined by IDS sensors to detect suspicious values [NFR2003].

As an example, let's use hex encoding to slightly modify URLs so that they would appear slightly different from our payload's packet signature to the human eye; however, they would have the exact same meaning to a web server. In this instance, if the "state" of the packet was unknown, then a very simple intrusion might be allowed.

Protocol analysis by itself is limited to examining a single request, although many attacks cannot be detected by looking at a single request. Usually an attack will involve a series of requests such as the fragmentation of packets which frequently occurs on any modern network wire today. The way these attacks can be detected is by using stateful characteristics to enhance our protocol analysis. In other words, an IDS sensor is able to remember data for the duration of the session and can correlate among several packets with multiple payloads that would otherwise be unable to detect [NFR2003].

NFR is able to detect known attacks, stealth attacks, anomalous behavior, Denial of Service floods (where the attacker uses several computers to "flood" the host computer with datagrams overwhelming it), brute force injuries (where programs are hit exhaustively with password crackers or other similar processes until memory is gone or program is vulnerable), and polymorphic shell attacks (where packets are encrypted along with a decryption device and when taken in by host, decrypter is run and "harmless text" changes into malicious code). NFR also maintains an up-to-date, round-the-clock rapid response team on staff, which monitors the Internet for new attack methods and exploits. This team compiles a database of these attack signatures, and immediately notifies the customer base and makes their data available for download. Also, new signatures can be instantly pushed to all sensors with a single button click [NFR2003].

An important part of NFR's strategy is the use of intelligent mapping [NFR2003]. Per NFR's documentation, an essential weakness of most NIDS products is lack of information. It is nearly impossible to perform true detailed analysis when individual packets are scanned with no knowledge whatsoever of the surrounding network. What if operating systems, services, and applications for specific IP addresses could be correlated with attacks? There would be a significant reduction of both false positives and negatives. If this were possible, alerts would be limited to that which was unprotected. For example, if there were a known exploit against an Apache server, hackers by the hundreds would attempt the crack anytime they believed an Apache instance to be present on a network they were attacking. However, if the System Administrator patches that exploit, then any alerts based on this attack, and there could be many, are now false.

Having surveyed the general methodologies of NIDS, let's take a closer look at SNORT, arguably one of the most successful and well-established NIDs on the market today.

CHAPTER FOUR

SNORT

SNORT is an open source and very successful NIDS. SNORT is essentially a combination of packet sniffer, packet logger, and network intrusion detection system. As shown in Figure 4.1, SNORT's architecture is composed of several main components, including preprocessors and several plug-ins, which make packet contents and alert mode systems much more manageable.

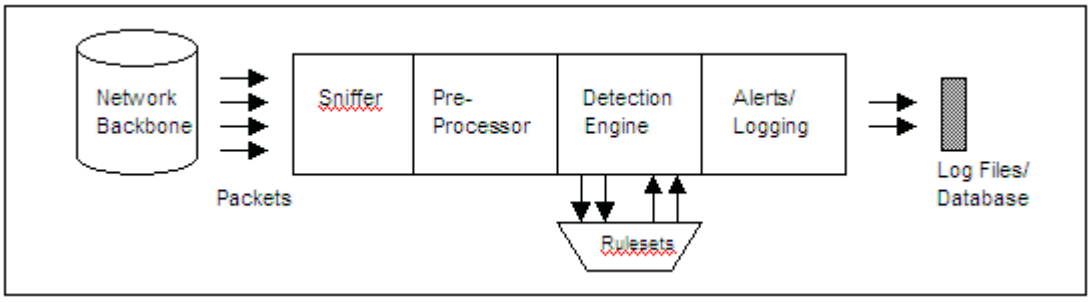

Figure 4.1 SNORT Architecture. Source [CBFP03], 34.

4.1 Subsystems of SNORT

4.1.1 Packet Sniffer

A packet sniffer is a device which makes it possible for an administrator to read network traffic. Just as telephone wiretaps are used by the police in their investigations, a packet or

network sniffer operates in much the same manner. A sniffer essentially allows an application to eavesdrop on network traffic. Because attacks generally come via the Internet, IP traffic is typically what is listened to[CBFP03].

4.1.2 Preprocessor

A preprocessor takes raw packets and checks them against minor applications usually plug-ins, which check for a certain type of behavior from the packet. An example of this would be a hacker trying to send 1000 small packets with the goal of deceiving the NIDS. A preprocessor would take 1000 packets, assemble them and translate them, and thus thwart the hacker's attempt. Essentially, packets are sniffed from raw network data and then passed to various preprocessors implemented by the system administrator, and then the preprocessor passes them to an encoding plug-in and finally to a detection engine [CBFP03].

4.1.3 Detection Engine

A detection engine is the primary part of the IDS in SNORT. Network data packets are captured by the sniffer and then handed to a preprocessor. The preprocessor may place them in a reassembly type plug-in, and after that the data packets are given to the detection engine. The detection engine analyzes these packets with a set of preprocessed rules. Hence, if there is a rule in the database and that rule corresponds to an attack, the detection engine has properly done its job and an alert is sent to the alert processor [CBFP03].

Each rule is contained in a group of rule sets, which are arranged according to certain categories such as buffer overflows, port scans, and Trojan horses. Each rule consists of two different parts.

1. The rule header—the type of action which will be taken based upon the type of network packet found. This includes source and destination IP addresses, as well as ports.
2. The rule options—the content in the packet that determines whether the packet in question matches the rule. [CBFP03]

4.1.4 Logging Component

Once the data has gone through the detection engine, it needs to be relayed on if a rule in the detection engine is matched and a warning or alert is triggered. These warning messages can be sent to log files, I/O screens network connections, UNIX sockets, Windows pop-ups, or stored in SQL databases [CBFP03].

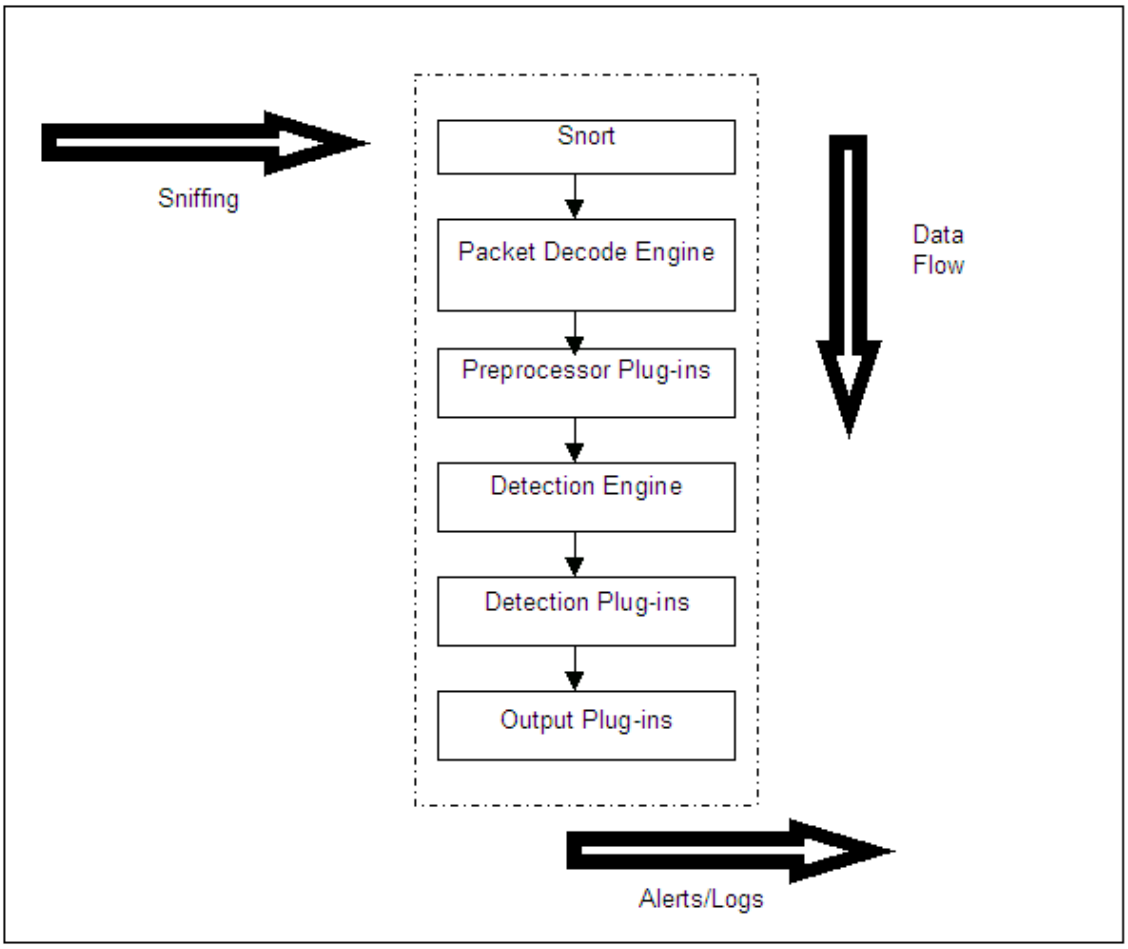

Figure 4.2 SNORT Mechanics Outline. Source [CBFP03], 95.

4.2 The Mechanics of SNORT

4.2.1 Sniffing

In order for SNORT to sniff data packets from network traffic, a network card is set in promiscuous mode. Usually, the default set up for a network card is to refuse or ignore data packets that are not destined for its particular MAC address. SNORT works by simply changing this behavior so the destination Mac address is not checked. In doing this, all the network packet traffic can be seen as it is placed on each hub sector. If packets are seen by a machine running SNORT, the network card, *unaware of destination Mac address*, will make this packet available to the Data-link layer. Once the packet has been captured and is present inside the Data-link layer, SNORT then uses the libpcap library to place the packet inside its packet decoder. This library is simply an extension of the TCPdump program [CBFP03].

When SNORT starts up, it uses a call inside SNORT.c to the libpcap library, which checks the interface and places the network card into promiscuous mode. Calling these functions and initializing these interfaces, SNORT enters the primary execution loop or pcap_loop. This endless loop receives packets from the network card device driver, which then calls the *ProcessPacket()* function. This function using the decode.c routine links into the Data-link layer [CBFP03].

4.2.2 Decoding packets

Once the data packets arriving from the network interface have been passed to the SNORT decode engine through the use of the libpcap library, the second stage (the operations loop) has been reached. At layer two, or the Data-link layer of the OSI model, SNORT needs to decode this raw packet.

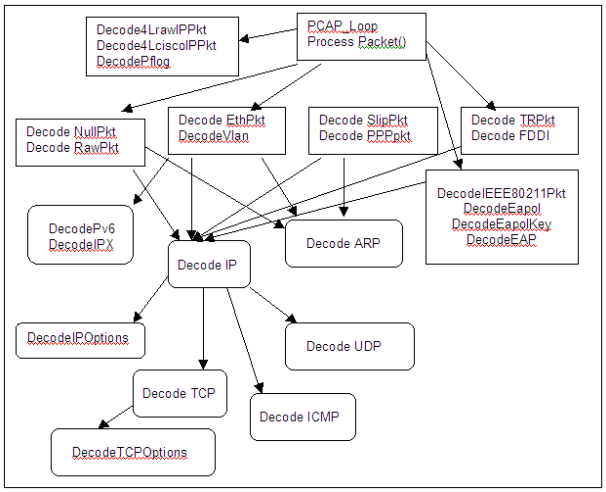

Figure 4.3 SNORT Decode Structure. Source [CBFP03], 104.

The *ProcessedPacket()* function now calls the *DecodeEthPkt()* function, which decodes each protocol. For instance, for a TCP/IP packet, the protocol is examined, and the *DecodeEthPkt()* function calls the *DecodeIP()* function . and finally the *Decode TCP()* function [CBFP03]. So now a decoded TCP packet exists and we now link it to the appropriate data structures and process it for detection. SNORT stores these packets in pointers and data structures held in memory [CBFP03].

4.2.3 Processing Packets

Having obtained network data packets and decoded them into several different protocols, they are now sent to the preprocessors in step three in the SNORT process. The introduction of preprocessors is to provide for a framework that will alert, drop, or modify packets before they reach SNORT's main detection engine [CBFP03].

Preprocessors can be divided into several tasks:

- Normalization—preprocesses packets into a standard format
- Fragmentation—dissects packets into pieces so they can fit more easily on network wire. Unfortunately, fragmentation can also be an easy way to bypass a pattern-based IDS. One hacking tool made famous for doing this is called *fragrouter* [HOL02]. Fragrouter fragmented network traffic into small bits and effectively evaded pattern matching systems. Since only portions of packets could be seen, there was no way for

a pattern-based IDS to make a match. Another downside of fragmentation, is that it has been used as a denial of service or DOS attack [HOL02]. If fragmented packets were extremely small, servers with valuable system resources had to reassemble them and this required a great amount of memory and processing time. Hackers realized that if network traffic was high enough and an IDS was occupied reassembling tiny fragments of packets, then the IDS could miss a single important packet which verified the signature [HOL02]. It was because of this danger that the frag2 preprocessor was created. This tool reassembles packets before they get to the SNORT engine, allowing signatures to be applied to full sessions and not just individual small packets. The preprocessor also alerts when fragmentation thresholds are reached[HOL02].

- State—The stream4 preprocessor allows SNORT to become stateful. In becoming stateful, SNORT can detect operating system fingerprinting techniques, and scans using *out-of-state* packets. It allows SNORT to keep its own state table and in this way, SNORT is aware of a full session [CBFP03].
- Port scans—port scans are regularly used to identify servers and the ports on the servers that are opened to services. Once a hacker finds ports which are opened, known exploits are sent to the services provided on those ports [CBFP03].

4.2.4 Rule Parsing and Detection

Once network data packets have been captured, they are decoded and placed into data structures in memory, and these data structures organize, filter, and decode the packet streams. The next part of the equation is the detection engine. SNORT rules are text based and stored in directories to be referred to by the SNORT binary files. At start up, the rule files are read by the *ParseRulesFile()* function and a three-dimensional link list is created [CBFP03].

4.3 SNORT Rules

SNORT has a particular setup of rules and how they are parsed. **[COD00].** The following is a rule from a sample rule set:

alert icmp !$HOME_NET any -> $HOME_NET any (msg:"IDS152 - PING BSD"; content: "|08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17|"; itype: 8;depth: 32;) [COD00]

The first part of the rule set is known as the header. It contains the action(alert), protocol(ICMP), source IP and destination IP and finally the port information. The second part is known as the rule option and contains information on what should be inspected in a packet to match to a signature in the SNORT rules. Remember, SNORT is essentially a packet sniffer and works by examining packets traveling through a specified network line or interface [COD00].

The rule above is looking for ICMP traffic not originating on the monitoring machine "!HOME_NET" destined for the monitoring machine "-> HOME_NET". The "depth" in the above rule is set to 32, which means SNORT will search 32 bytes into each packet looking for the specified "content". If matching content is discovered, SNORT will generate an "alert"[COD00].

alert tcp !$HOME_NET any -> $HOME_NET 31337 (msg:"BACKDOOR ATTEMPT-Backorifice";flags:S;) [COD00]

1. This rule is using the "alert" option.
2. Looking for 'tcp' traffic traffic originating from outside the home network "!$HOME_NET" and destined to the monitoring machine "-> $HOME_NET" on a specified port "31337"[COD00].
3. The flags option in this rule is looking for a SYN.

This rule is looking for any traffic originating from outside our network and attempting to connect to port 31337[COD00].

```
alert tcp !$HOME_NET any -> $HOME_NET 21
(msg:"SCAN-SATAN-FTPcheck";flags:PA; content:"pass -
satan";)[COD00]
```

1. This rule is using the "alert" option[COD00].

2. TCP flags are set at "P"ush and "A"ck looking at port 21(ftp)

3. Data sent from client contains "pass -satan"

This is a portscan from the program Satan trying to connect on ftp port[COD00].

4.4 Packet Matching

The processed rules files exist in a SNORT 3-D linked list. There are five separate chains of rules [CBFP03]:

1. Activation—alert and turn on another dynamic rule
2. Dynamic—log the traffic
3. Alert—and then logs the packet
4. Pass—ignore this packet
5. Log—log the traffic, do not alert

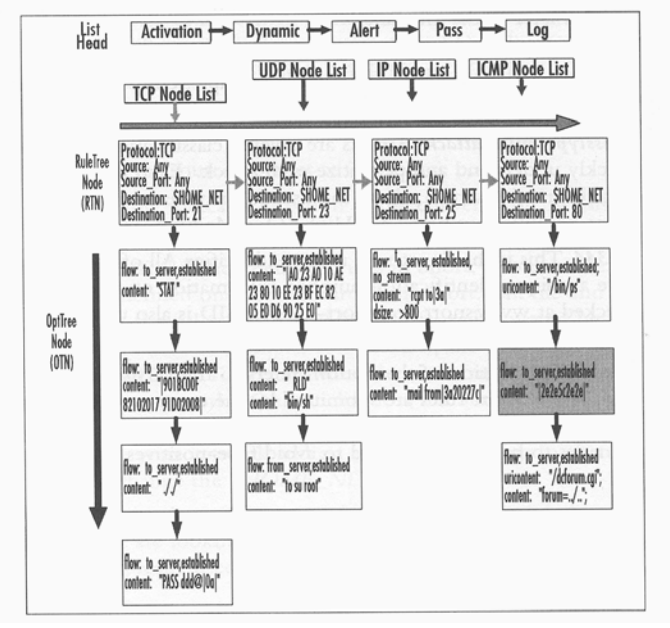

Figure 4.4 SNORT Rule Tree. Source [CBFP03], 118.

For each of the five chains there are separate lists by protocol. [CBFP03]

- TCP
- UDP
- ICMP
- IP

For each of the protocol lists there are rule options. These are referred to as the option tree nodes. Upon initialization, SNORT will read the rules files and populate the linked lists. Once the linked lists are built, the lists must be searched to find a match to the pattern. When the packets arrive at the detection engine, SNORT proceeds by the following order: Activation, Dynamic, Alert, Pass, and Log [CBFP03]. Then within each of these rule headers the Rule Tree Nodes (RTNs) and the Option Tree Nodes (OTNs) will be checked. The search method is dependent on the protocol of the packet. For example, if a TCP packet were in question, SNORT would start at the TCP node list and go to the RTNs where the packet would be checked for:

- Source IP address
- Destination IP address
- Source Port
- Destination Port [CBFP03]

If SNORT finds a match with the information in the packet, then the search continues looking for a match inside the OTNs. The algorithm used is the Boyer-Moore Fast String Searching Algorithm which is the fastest and most efficient algorithm for finding matching patterns in a string. Finally, if a match is found, the structure is exited using fast exit strategy and an alert is made to the output format [CBFP03].

SNORT has definitely evolved over the years and lessons involving speed and accuracy have been gleaned from changes made, we will discuss these in the next section.

4.5 SNORT—Evolution

SNORT was originally intended to be a simple packet sniffer. It was written by Marty Roesch in November of 1998, and at the time it was a Linux only packet sniffer called APE. Not satisfied, Roesch wanted a sniffer that also would work on multiple OS's, use a hex dump

payload dump, (which did not exist in TCPdump at that time) and display all the network packets the same way, (which also did not exist in TCPdump then). At this time, SNORT was composed of about 1600 lines of code. SNORT's first signature based analysis came in January of 1999, marking the first time SNORT could be used for intrusion detection. SNORT's main strengths are that it is lightweight, fast, and rudimental in operation [CBFP03].

The latest version of SNORT contains approximately 75,000 lines of code and contains a completely new architecture. Improvements include a protocol flow analyzer, a new detection engine, an enhanced rules language and several performance enhancements, which gave SNORT 2.0 eighteen times the processing speed of SNORT 1.9 [CBFP03].

The protocol flow analyzer can classify network application protocols into client and server data flows. Once an application protocol is classified into client flow and/or a server flow, it can provide SNORT with useful knowledge about the type of inspections and the regions of the inspections which the protocol flow should be monitoring [CBFP03].

The new detection engine is broken into three parts, the **rule optimizer,** the **multi-rule search engine,** and the **event selector**. The **rule optimizer** uses a set based methodology for managing rules and applies them to network traffic. Rule subsets are based on unique rules and packet parameters using classification schemes. This allows the entire SNORT rule set to be divided into many smaller subsets which enhances the search pattern in that all applicable rules are tested against each packet and ensures that no other rules could possibly match the packet.

The **multi-rule search engine** is broken into three searches based on rule properties.

1. Protocol field search—allows a rule to specify a particular field in a protocol to search
2. Generic content search—allows a rule to specify a generic byte set to match against the payload
3. Packet anomaly search—allows a rule to specify characteristics of the packet or packet header that is cause for alarm

The **event selector** allows SNORT to track every occurrence of every rule mentioned within a packet.

With each of these modifications there is a consistency to the developments. All improvements enhanced the speed of search by either a faster lookup of matches, a way to ignore completely bogus traffic, rule optimization, or enhancements to 'state' knowledge. [SNORT]

CHAPTER FIVE

RULE BASED ARTIFICIAL INTELLIGENCE

Artificial intelligence or AI as it is known, has been around for quite some time. It is a field of computer science that attempts to mimic or copy human-type thinking and action [OL00]. Unlike simple processing of information with selection statements and working memory, artificial intelligence attempts to replicate thought processes such as reasoning, intuition, learning from past trial and error, and generalizations [FR00].

Although difficult, some success in replication of human intelligence has been achieved by what are known as expert systems. Usually these systems reside on very powerful machines operating at extremely high speeds and the programs themselves are incredibly complex. Expert systems are actually in a class of artificial intelligence known as rule-based systems [FR00].

Rule-based systems work by representing knowledge in much the same way as a decision tree. A decision tree has branches based on situations that lead you to a final node on the tree which is hopefully the specific, correct answer to the given situation. Knowledge is usually represented inside a computer in a declarative, static construct. This could be a database, a vector of information or results, or any other structured data source. Instead of representing knowledge as a group of objects which are true by default, rule-based systems use groups of rules telling you the user what you should do or what should be true in different situations.

5.1 Rule-Based Systems Mechanics

Rule-based systems typically use a set of positions or assertions which form a knowledge base or 'working memory'. Using a set of rules which specifies how to maneuver on this knowledge base or assertions set, a rule-based system now is able to give a dynamic response to a user’s question on the situation. Although sounding complex, rule-based systems are typically

very simple, usually little more than sets of if-then statements. These systems generally use some type of interpreter, able to control the application of these rules given an input of situational facts [FR00]. The reason that some rule-based systems are called expert systems is that the knowledge of an expert is encoded into the rules set. If given the same input data, and using an expert to create the systems, it follows logically that an expert system should perform in a similar manner to the expert himself.

Rule-based systems are very versatile in that they are a relatively simple model which is easily adapted to any given number of situations. As with any computer modeled artificial intelligence, rule-based systems share various strengths and weaknesses. Rule-based systems are usually only applicable to problems or situations where any and all of the knowledge in the problem can be written in the form of selection rules and also where the problem area is not extremely large. If there exists too many selection type rules the system becomes a nightmare to maintain and suffers huge processing performance hits [FR00].

In designing or creating a rule-based system for given dynamic situations you must possess the following attributes.

- The relevant group of facts or precepts which represent your knowledge base or working memory.
- The set of selection rules which include any and all actions which could be taken based on a given situation. It is important to note that only relevant actions or rules should be included and nothing unneeded which could affect system performance.
- A guaranteed Boolean termination situation that will positively end the processing in the system. [FR00]

5.2 Classes of Rule-Based Systems

There are two classes of rule-based artificial intelligence systems: forward chaining systems, and backward chaining systems. Forward chaining systems start with initial facts and use rules to find conclusions or take actions given the initial facts [FR00].

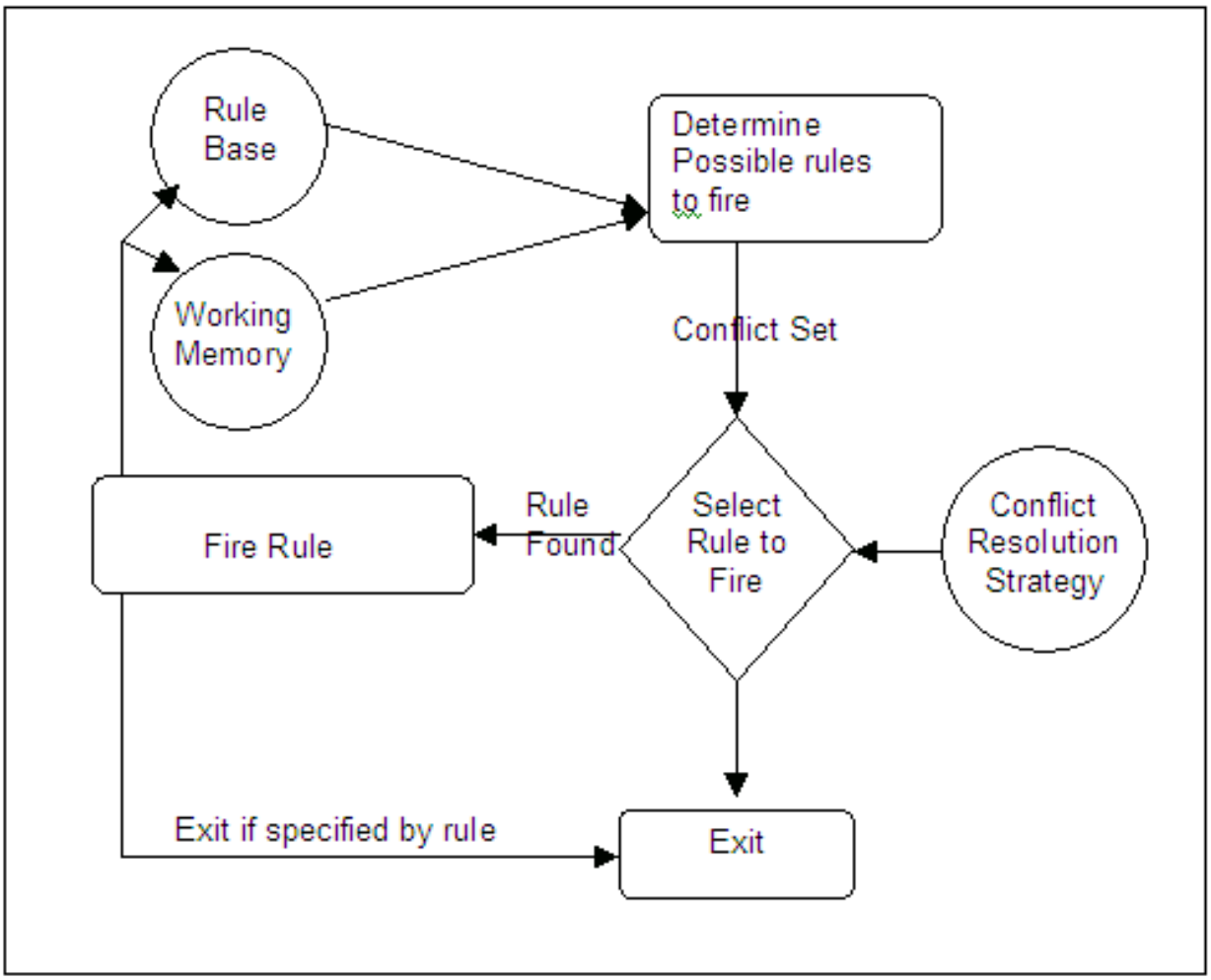

Figure 5.1 Forward Chaining Procedure. Source: [FR00].

Backward chaining systems start with the answer to the situation itself and look for rules to conclude that the answer to the situation is the correct one [FR00].

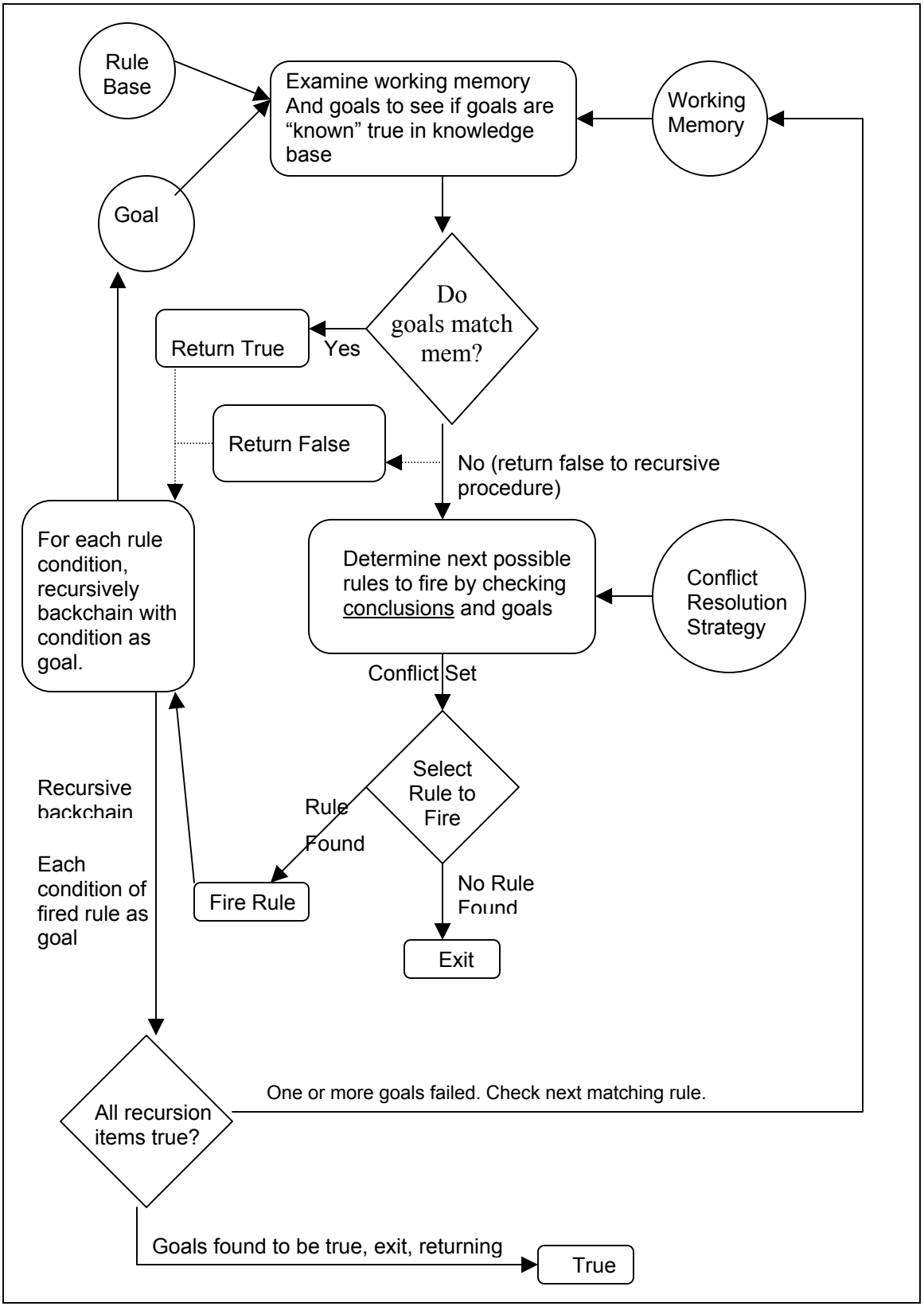

Figure 5.2 Backward Chaining Procedure. Source [FR00].

Usually forward chaining systems are data or knowledge driven and backward chaining systems are goal or answer driven.

In the designing of rule-based systems several classes of reasoning have arisen as the forefront of modern rule-based systems. These classes attempt to mimic or affect the ways that humans think about solving problems.

1. **Derivation or deduction rule-based systems**—in each rule the system tries to express the knowledge that if a set of statements is true another set of statements must also be true, this is known as a logical implication.
2. **Transformation rule-based**—one rule in one knowledge base if true has a truth relationship to a rule in another knowledge base.
3. **Integrity constraints** rules of the form 'it must be true' or ' if it is not true that ---- then we have an error'
4. **Reaction rules** these involve notions of actions not just inference when a rule applies. [FR00]

A rule-based system is procedural, it begins with data expressed in if-then rules and examines the knowledge base which has previously been defined. All the rule conditions are examined and they determine what is known as a 'conflict set'. Out of this conflict set, a rule is chosen based on the systems conflict resolution strategy. Once a rule is chosen the action in the rules THEN clause is fired off. This action can dynamically change the knowledge base, the rule-base, or some other predefined situational proposition.

The system must be able to terminate and rules are fired off until all conditions are satisfied or possibly a final rule is fired part of whose action is to terminate the program. The rule that is fired has been chosen by the systems resolution strategy, and the strategy that is chosen depends on the preference of the situation. The following is a list of different strategies:

- First applicable—if rules are ordered then fire the first applicable rule.
- Random—once the conflict set has been established the system chooses a single random rule from the set to fire
- Most specific—once conflict set has been established fire the rule with the most conditions satisfied
- Least recently used—a timestamp which marks the last time rules are used

- Best rule—rules are given weights based on how much they are considered over alternatives and the rule that is most preferable or ‘weighty’ is chosen.

CHAPTER SIX

ANALYSIS OF SNORT PERFORMANCE

As with all signature based detection, SNORT is primarily used for active auditing. Signature based detection implies a predefined entry in a rule base for what an attack looks like. Once the signature is in the database or list, then the network monitoring software looks for that specific signature. Although an excellent tool, SNORT has three major downfalls:

- Packet dropping
- False Positive Alerts
- False Negative alerts

Because of speed issues with a network, SNORT may not pick up all packets. Other factors which can affect SNORT in this way are the speed of the promiscuous interface and the stack implementation of the operating system. It is important to note that SNORT is able to be overrun with packet flooding which then makes the detection of intrusions more difficult.

False positives occur when SNORT sends alerts when it shouldn't, in other words a false alarm. This can happen for various reasons. Some of these include:

- Placement of SNORT outside of the security perimeter—SNORT would receive DNS scans, web proxy scans and other various benign informational network that would cause overload for the system administrator.
- Site Policy allowing activity that causes IDS alarms—For instance, using the default setting for SNORT which would increase the data inflow to an unmanageable level.
- Lack of site awareness in the IDS—Not being aware of services running on hosts, such as IIS attacks on Apache web servers could lead to false alarms.

False Negatives occur because of any attack not matching a signature in the ‘known attack’ database. This can happen because of poor rule design, encrypted or otherwise cleverly

disguised traffic, or simply because the attack is new and has never been signature matched.

6.1 Code-Red Detection

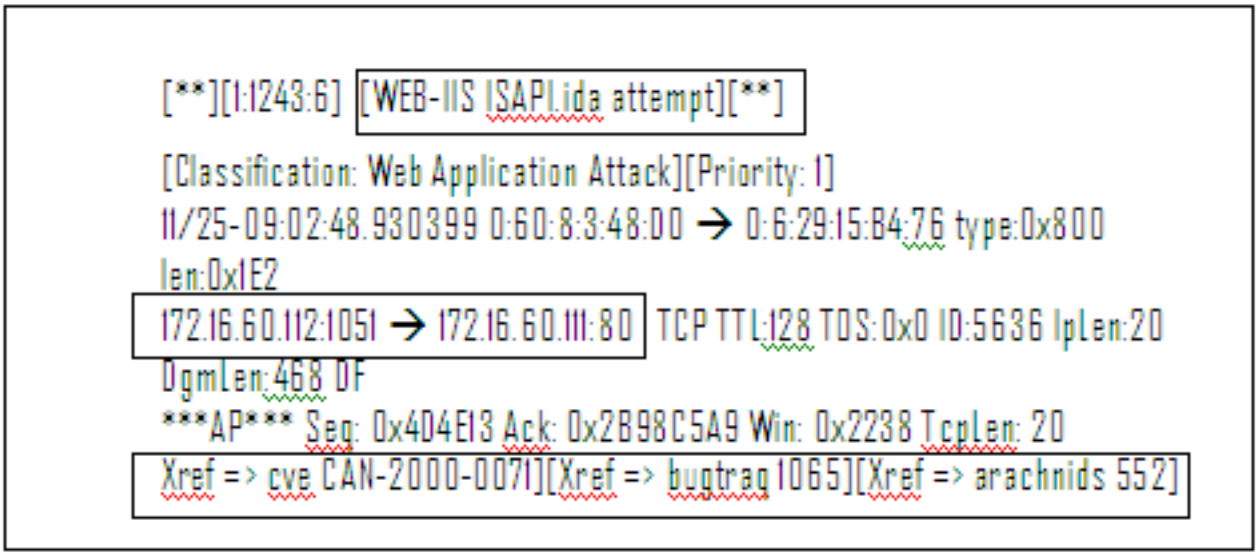

Figure 6.1 SNORT Alert After Code-Red Attack. Source [CBFP03], 15.

If configured correctly, the following SNORT alert would be sent to the system administrator. This alert contains the classification of the attack, the IP addresses of the parties involved, the protocol, and references to find information about the attack.

A default log is created for every attack found by SNORT. The sample log entry below was generated by the attack in Figure 6.1. The signature for the Code Red worm includes a truncated display of the original 254 “N” characters whose only significance was to overflow the buffer.

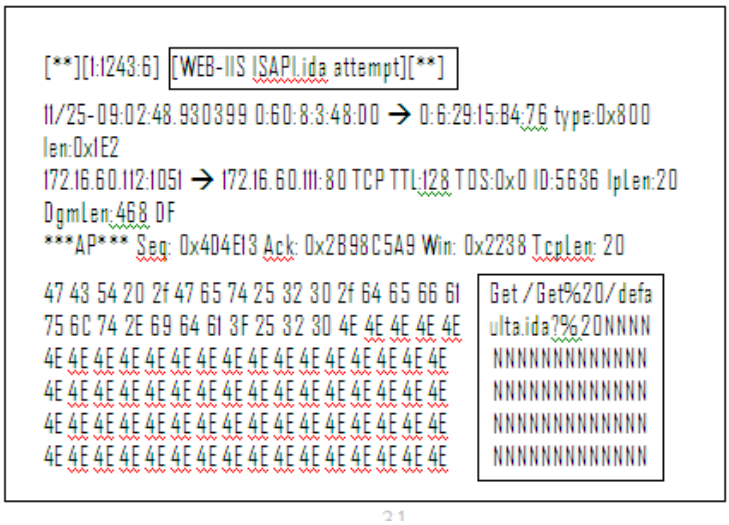

Figure 6.2 SNORT Log After Code-Red Attack. Source [CBFP03], 15.

6.2 NIMDA Detection

The following is the SNORT alert which was sent after finding a NIMDA worm. As in the previous example, Priority, Attacker IP, Victim IP, as well as a reference to the *CERT advisory CA-2001-26* have been included in the alert.

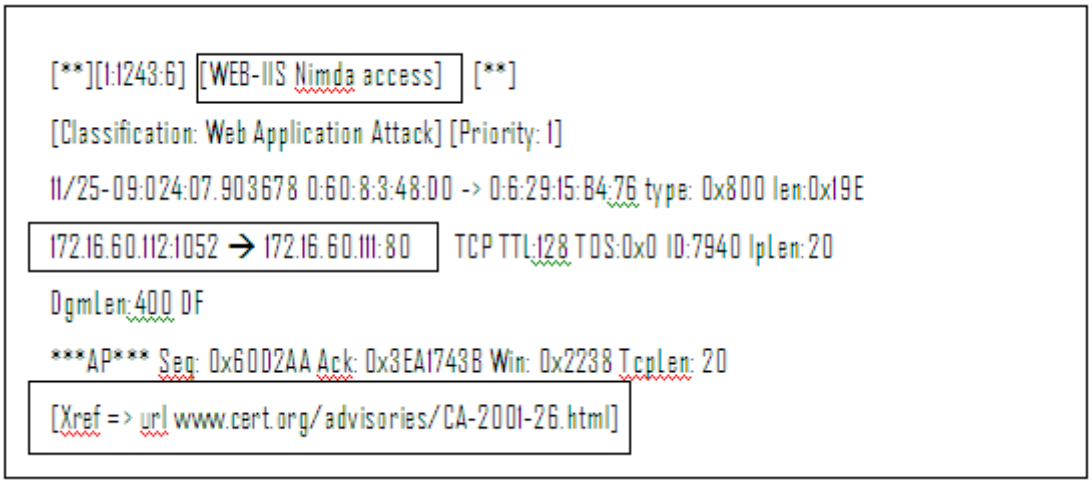

Figure 6.3 NIMDA Alert. Source [CBFP03], 15.

Again, a default log is created for the attack reported by SNORT. The sample log entry below was generated by the attack in Figure 6.3. This entry in the highlighted box shows the 'scripts/root.exe' signature. This is from a file left behind by the CodeRedII worm and resulted by copying the Windows CMD.exe as root.exe. SNORT detected NIMDA's attempt to use this backdoor and generated both the above alert and this log.

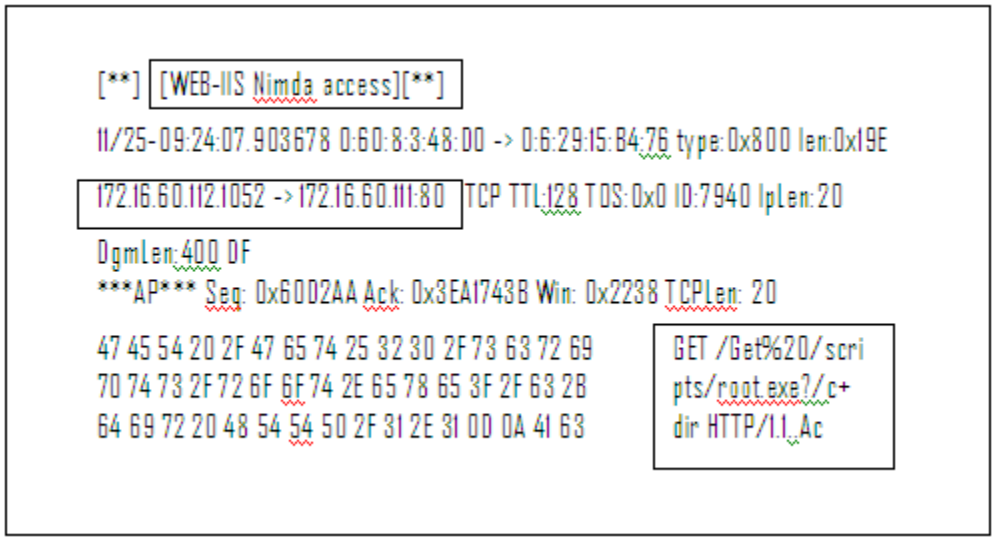

Figure 6.4 NIMDA Log. Source [CBFP03], 15.

SNORT uses the fastest and most well developed signature based engine today. SNORT was actually written to take advantage of a highly modularized design. It has both preprocessors and post processors which enable SNORT to filter and categorize and output data in several different ways. Also, because SNORT is open source, its signature database is updated often and it is very simple to update. The reliability performance of signature based IDS mechanics is based on attacking packets having patterns which can be matched against a database. Essentially, this means that SNORT is a sniffer combined with a high speed string matcher.

SNORT Weaknesses:

1. Have any knowledge of, or identify a previous intrusion or intruder attempt relative to, the current time because SNORT simply looks at the packet involved in the given scan and not in a historical context.
2. Be aware that a current intrusion attempt was made by the same intruder found by a different sensor because it does not compare one attack with another to determine a pattern.
3. Know the network or system setup or even be able to determine if a given alert is pertinent to the system because SNORT is only a packet sniffer and string matcher without any knowledge of the network itself.
4. Know that the network it protects is under a denial of service (DOS) attack, while it may be able to detect a type of signature it is completely unaware of other attempts made because of the limitations of its working context.
5. Understand which hosts are more vulnerable to attackers than others because, once again, it has no outside knowledge of the network which it services.

CHAPTER SEVEN

PROBLEMS WITH SNORT

The White paper, *Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection* identifies problems with network ID systems [PTA02]. The first theory that Ptacek proposes is that there is insufficient information available in packets read off the wire to correctly reconstruct what is occurring inside complex protocol transactions. Secondly, he proposes that ID systems are inherently vulnerable to denial of service attacks. The first of these problems obviously reduces the accuracy of the system, but the second jeopardizes its very availability [PTA02].

Network IDS work by capturing packets off of a wire and determining what is happening on the machines to which the packets are addressed. Granted, a packet in its simple form isn't nearly as significant to the system as the packet behaviors on the machine receiving that information and processing it. In essence, network IDS try to predict the behaviors of the network machines. They do this by trying to base the packets behavior on the machines the packets will be exchanged on [PTA02].

One obvious problem with this technique, as previously identified as weaknesses # 3 and # 5, is that passive network monitors cannot possibly know if the machine on the network they are trying to protect is even going to see a packet [PTA02]. Or, more importantly, will the machine process that packet in the expected manner? A number of issues exist which take the meaning of a packet described by the IDS as ambiguous.

A network IDS like SNORT will most likely be operating on an entirely different machine from the network systems on which it is eavesdropping. Because of this, there may be inconsistencies between SNORT and the machines or networks that it watches. For instance a machine might be down at the time an attacking packet is intercepted.

Even if an IDS knows the operating system of every machine on the network runs, it still cannot tell just by looking at one packet whether a machine will accept it. What this means is that an IDS as often as not, cannot determine the implications of a packet merely by examining it out of context (weakness #1, #3, #4, #5). It needs to have a great deal of knowledge about the network and the networking behavior of the end systems that it is watching. It also needs to know the traffic conditions of the network segments. With a network IDS, there is no simple way of informing itself about this. It truly obtains all the information and *knowledge* it has from packet capture.

Improvement upon IDS as based on this paper would require the IDS to have more knowledge with which to make decisions. Most noise and false positives occur inside the network because of the lack of site awareness by the IDS [PTA02]. Most IDS, including SNORT, truly cannot determine if targeted systems are open to the attacks launched against them. Some IDS are able to determine host vulnerability, yet it is possible for these systems to still log totally benign attacks as dangerous and improperly raise alerts based on incorrect administration setup.

Often false positives occur during deployments of software or custom applications are running and these confuse IDS into thinking that an alert has taken place. A good example of this are load-balancing devices, which send probing packets to systems to test their responsiveness. To a network-ignorant IDS, these could easily be marked as reconnaissance probes but these probes often provide very little useful information for the system administrator. Administrators often have the choice in using an IDS like SNORT to turn off signatures to end the false positives, whereas other administrators may leave these turned on, fearful that they will miss a true attack [PTA02].

In a nutshell, the main difficulty with modern NIDS is that they need to be smarter. Currently, most NIDS only concern themselves with analyzing packets and obtaining signatures managed with predefined rules. This is an excellent beginning, but there are ways to find out so much more information to determine much more accurately if an attack truly has taken place.

Understandably, companies are taking the best that technology has to offer and implementing it in such a way as to maximize their potential returns. But what about the big picture? Packet examination is great, but imagine how much more useful it would be if combined with the larger picture. The way things stand right now is the equivalent of a circus sideshow.

The sideshow gives the spectator a BB gun to take potshots at plates set up on a table. A blindfolded clown tries to dodge in front of the plates so that they are protected. In this analogy, the person with a rifle is the hacker and the plates are individual hosts on a network. The clown is the NIDS which is trying to stop the shots. But just like a clown who has no idea if there is even a plate behind him, the NIDS has no understanding of the environment. Nor does it understand the hosts on the environment or the services set up on each.

A practical example of this would be if SNORT detected a high-priority alert on a Microsoft IIS server attack. SNORT logs the attack and then sends an alert to the output screen or to an ACID database, to be analyzed at a later time. Now SNORT has done its job perfectly well, exactly the way it was programmed to. But imagine that this occurs on a large-scale network with several thousand nodes. Each time, SNORT does its job. Packets are captured, and packets are logged, and system administrators are alerted. The problem is though that there are no Microsoft IIS servers on the entire network. There are, however, thousands of logs which have been generated, and probably thousands of alerts, and which tell the system administrator that a high-priority vulnerability exists and was attacked many times over in a given period. SNORT has done nothing wrong, according to its function. Unfortunately because of the reality of this network environment, SNORT has actually done a disservice, because now thousands of alerts, which are irrelevant to network security but still have to be sifted through by a human being.

Another example, using the behavior pattern of a typical hacker, would be the use of port scanning. SNORT, if set up outside a firewall, would be able to pick up a broken TCP/IP packet used for sniffing ports. SNORT would log this and then possibly alert the system administrator of this activity. Later, after the hacker has done his portscan routine, he might initiate a buffer overflow attack on an application with the open port which he has detected. SNORT hopefully would pick up this attack, log it, and then send an alert if set up properly to whomever is monitoring the system.

Unfortunately, SNORT has no way of knowing that a port scan by the very same IP address was previously detected within the hour, or possibly the last week or month. Wouldn't it be much more beneficial if SNORT had a high-priority focus on that IP number being a probable attacker? Instead, SNORT has no context each time it detects an attack. The system administrator

must use his own knowledge and great amounts of time to discover what would have been obvious to a human being watching the packet flow as it occurred.

These two examples clearly illustrate that the NIDS must be given more information, and made "smarter", if we hope to make any kind of gains in the field of computer security. First and foremost, a list or database of attackers or ex-attackers should be compiled to aid the system administrator. This database should have statistical analysis performed at regular intervals in order to glean pertinent information and reduce the memory stress on the server which contains the database. It is important to note that stealth attacks committed by hackers can last up to a year.

Secondly, a type of snapshot should exist so that the NIDS can compare or correlate attacks and vulnerabilities based on packets received and host information on the network itself. Without this capability, the ingeniousness of attacks will overwhelm these systems in a short amount of time. Also other tools, such as file checkers like *Tripwire*, could be used in a similar capacity as long as the NIDS has a centralized logical place for network intrusion analysis.

As recently as a few months ago, IDS technology has garnered some negative press. "IDS as a security technology is going to disappear," says Richard Stiennon, a research director with the *Gartner Group*. [HG03] Stiennon claims that IDS, based on their expense, do not provide enough value to organizations. The article goes on to report that companies complain that systems which they deployed generate more alarms than they could possibly investigate, and worst of all that those alarms are being generated when no attack underway.

Martin Roesch, the founder and developer of SNORT, concedes that IDS—including SNORT—have their failings, including the amount of noise, false negatives, and false positives that occur with the installation of any intrusion detection system. To rectify that, Sourcefire, a company which Roesch heads as technology officer, will be creating a real-time network awareness appliance to aid detection technology. This will be specifically targeted towards making IDS, particularly SNORT, more aware of the network environment.

According to Mcaffee Software, the primary research goal for their IDS system is accuracy, or the reduction of false negatives and positives. This is also echoed by Philip Hollows in the article " IDS Is Dead—Long Live IDS?" which states that "IDS have long been derided as difficult to manage, creating many false positives and negatives, which is one of the reasons that security event management solutions evolved -- to make IDS both more manageable and more

effective". [HP03] He goes on to state that the key problem is that most packet sniffing solutions are context-free. "They have no idea whether an attack is relevant and the volume of events that they produce tend to hide the dangerous attacks in low risk noise".

Unfortunately for IDS, in addition to accuracy, the system administrator also suffers a more insidious psychological effect. The article "The Effect of Crying Wolf", a UNIX security paper from 2001, states "when a car alarm goes off repeatedly, the thought is not one of concern but irritation. When you hear a car alarm, you do not take action to stop a car theft in progress, you just get annoyed that someone has an oversensitive car alarm…When false positives crop up in computer security measures, they can have a very detrimental effect on overall security". [US01]

The underlying point is that the more false alerts, the less useful the alerts are for security policy. Reducing false alerts is very difficult and it takes a clear understanding of network specifics to adequately understand an attack. This is the heart of the problem we are addressing in this paper.

An improved and modified SNORT should contain some sort of contextual reference. In other words, it should be made more intelligent. It should be able to recognize vulnerabilities in hosts within its surveyed network. It should also be able to distinguish users from attackers. It could do this simply by maintaining a list of previous IP numbers which attempted attacks on hosts in the network it was protecting. Vulnerabilities would be mapped in much the same manner, and different hosts would be listed along with their weaknesses in a centralized data structure. In essence, what we are proposing, is a centralized framework used as a information shell around SNORT to assist in making decisions about improper usage and attacks on hosts.

CHAPTER EIGHT

DESIGN OF A SMARTER SNORT

This thesis proposes a solution for the problem of false negatives, false positives, and network noise through the use of network and intruder knowledge awareness. From SNORT documentation, it has been clearly identified that there is simply not enough information present inside the SNORT engine to make any knowledgeable assessment of a true attack. Obviously packet analysis is necessary to detect attacks, but an additional level of information and decision processing is required.

The most logical improvement would be the addition of a knowledge base which would include not only a comprehensive history of attacks, but necessary network vulnerability information as well. Although SNORT by itself was a successful, highly rated intrusion detection device, it offered no systematic attack analysis by itself. To rectify this situation, a completely separate, parallel knowledge base would work in tandem with the SNORT, which would still function as the primary attack detector.

Using the SNORT detection engine as an input, we propose a rule-based tool working in parallel, which would sift through the SNORT alerts with artificial intelligence and act as a security assistant for the system administrator. This design will be referred to as the **Rule**-based **Alert Management Information Tool** (**RAMIT**).

8.1 Rule-based Alert Management Information Tool (RAMIT).

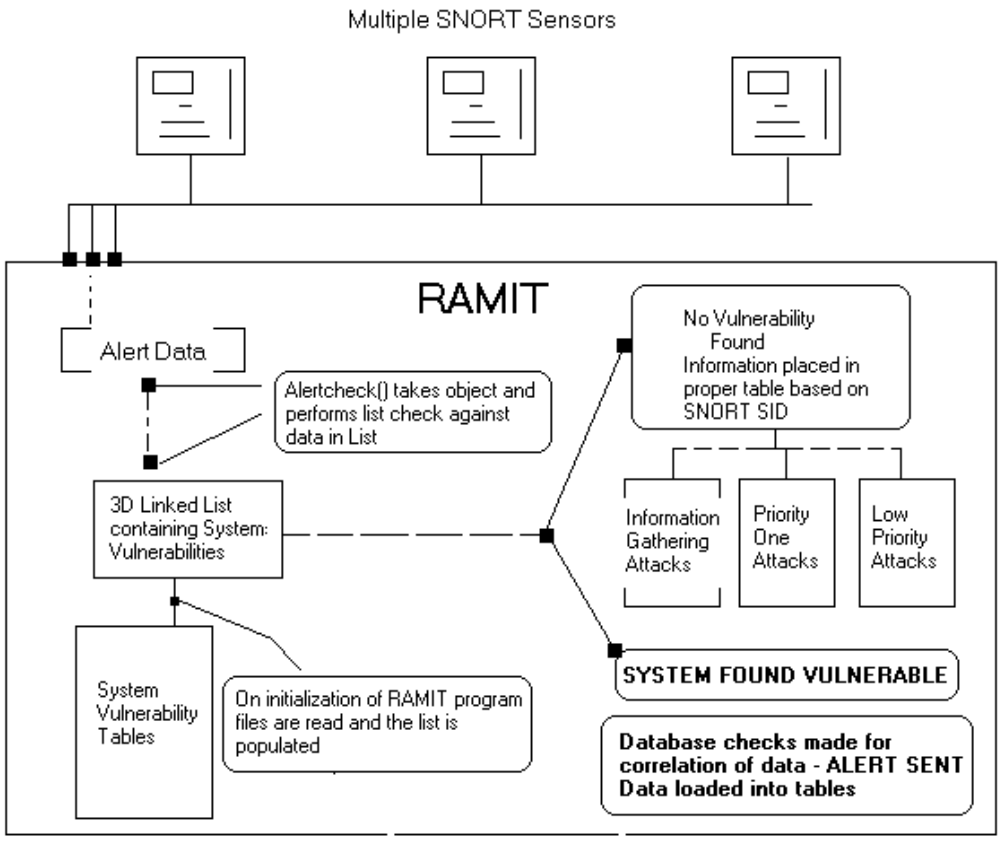

Figure 8.1 Rule-based Management Information Tool (RAMIT)

Because SNORT is completely limited to packet analysis, and contains no history of prior intrusions, the device is very limited in its functionality. RAMIT, however, would maintain a real-time accessible network and an intruder database based on previous activity within a connected environment. Once SNORT sends an alert with information containing IP of attacker, IP of victim, timestamp, etc., RAMIT would accept this as a real-time parameter and, based upon the intelligent rule-set, then make a decision as to whether or not to notify the system administrator of a possible threat.

Therefore, RAMIT could be seen as a first-line information, correlation defense processor, doing much of the routine work that would ordinarily occupy the time of the system administrator. Not only would RAMIT maintain separate databases of intruders and network host vulnerability information, it is also designed to correlate multiple SNORT sensors in real-time. This would give it the ability to assess and gain knowledge without a host having to be attacked. All information would be centralized and sent through encrypted lines to a host containing the RAMIT process.

RAMIT's design completely relies upon SNORT sensors for information, and once the sensor has sent an alert parameter, then SNORT's interaction with RAMIT is at an end. SNORT's role would now be to act primarily as a rule checker and as a packet sniffer. Once SNORT makes a match, finding a rule to match the packet description, it would then send this information to RAMIT for processing. SNORT is now free to continue finding new alerts without having to divert resources to alert the system administrator. As designed, RAMIT would be in a much better position knowledge-wise to make a recommendation to the system administrator based on a rule. RAMIT uses a predefined rule-set for determinations of vulnerabilities to all the network hosts in its environment, and it is this knowledge-based linked list which would determine if an alert represents an actual threat.

RAMIT is designed to employ a secondary database to check the past history of the IP number in the IP attacker field of the alert parameter. The type of alert sent by RAMIT is determined by the IP number and its position in the three intruder databases. These three databases catalog port scanning threats, priority one attacks, and lower-level attacks. Once this information has been analyzed, RAMIT would then have the necessary knowledge to make a proper decision as to whether to notify the appropriate personnel.

8.2 Overview of RAMIT Operations

As previously discussed, most NIDS (including SNORT) produce too much data in the form of alerts without the benefit of the knowledge needed to make a decision as to how to act on that alert. Our design proposes four major areas from which extra knowledge could be

gleaned and used to increase the accuracy of an alert response and in determining potential vulnerabilities in our protected network.

1. Multiple SNORT sensor input: RAMIT would be able to receive data from multiple inputs in real-time.
2. Network awareness: Whereas most NIDS have no knowledge of the surrounding network, ports open, and services running, our solution design will give listed vulnerabilities to each host on the network which are then checked against each incoming SNORT sensor parameter working as a parallel structure in real-time.
3. Intruder awareness: RAMIT would retain all SNORT logs to refer to a later time, and through these we can judge whether SNORT inputs are coming from previously identified intruders or hackers.
4. Centralized server: All information coming into the alert server is processed on the server and would serve as the system administrator's main tool for alert response and policy.

8.3 RAMIT Pattern

RAMIT's design designates a centralized machine to accept the 'alert parameters' of input from SNORT sensors which are located throughout the monitored network. This machine would contain an artificial intelligence rule-based system which performs analysis on the input parameters in much the same way a human would. As 'alert parameters' are taken into our database, knowledge of attacks presented would increase.

These parameters contain data concerning an attack and are duly recorded. This data would then be matched with known vulnerabilities on each host. If a host is found to be vulnerable, then the program would make system SQL calls to judge whether the attacking IP has attacked before or if a scan has been made of the system. Also, a check would be performed regarding the seriousness of the attacks.

Once this information has been processed, a real-time alert would be presented in HTML to the system administrator. Although the machine running the system must be dynamic in terms

of resources and output, it would allow excellent scalability for SNORT since all alerting and processing of the alerts would now be performed by our system, letting SNORT sniff the network and process datagrams.

8.4 RAMIT Input

As stated in the SNORT documentation, the most useful set-up for SNORT sensors includes one inside the router, one inside the demilitarized zone, and one inside the firewall. Additionally, a SNORT sensor should be placed inside each subnet to be protected.

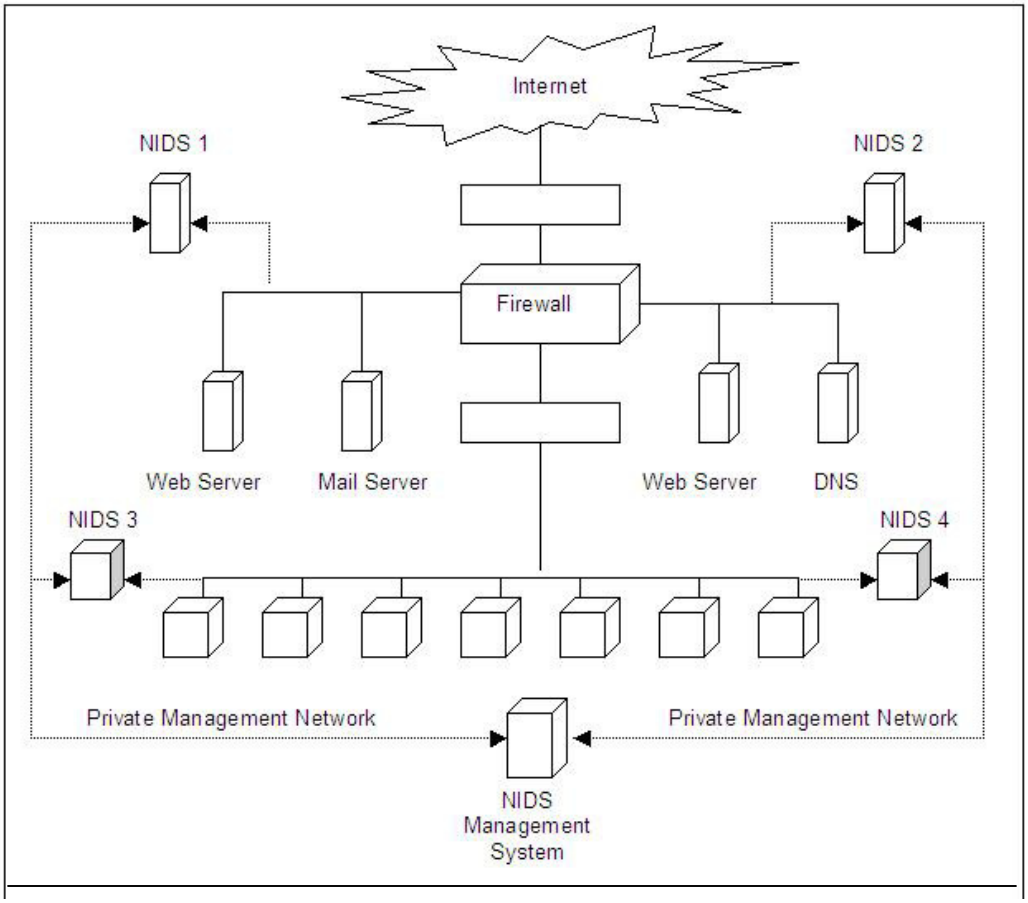

Figure 8.2 Distributed Intrusion Detection System Schema. Source [CBFP03], 7.

Any time that SNORT detects a signature match on any sensor, that information would be logged on the local machine. As RAMIT would now be determining real-time alerts, the alert mechanism inside SNORT can be disabled or removed. Instead, the information that SNORT sends for an alert would be modified somewhat. A text-delimited object will be sent over encrypted lines to the centralized RAMIT server. The information inside the text-delimited parameter will contain:

- SID - SNORT SIGNATURE ID Number
- TIMESTAMP – Time the attack took place
- PROTOCOL - TCP, UDP, ICMP
- PRIORITY - 1, Information Gathering, etc.
- PORT
- SOURCE IP
- DESTINATION IP
- MESSAGE

An example of the parameter the SNORT sensor sends would look like the following:

359,00:38:02,TCP,, 21,213.2.13.2,196.23.12.8,FTP Satan Scan

8.5 RAMIT Functions

The input parameter, taken from the SNORT sensor, signals the mechanisms of RAMIT to initiate. The following functions follow the timeline from receiving the SNORT parameter to the actual alert of the system administrator.

8.5.1 `GetParam()`

We will assume that SNORT has found a signature and has sent it to the centralized RAMIT server. RAMIT would receive this parameter over a secure AES 256 bit connection with the function `GetParam()`. Upon initialization of the RAMIT program, a parameter would be requested for how many SNORT sensors are active. Beginning with TCP port 60223,

`GetParam()` monitors for information received on this port and each subsequent incremental port thereafter. For instance, if three sensors were active, `GetParam()` would listen on ports 60223, 60224, and 60225. Our centralized server containing RAMIT would listen for incoming connections from SNORT sensors on these ports, and once a parameter was received, the vulnerability assessment function `Alertcheck()` would be called.

8.5.2 `Alertcheck()`

Once SNORT sends the attack information to the Alert monitoring tool, the object would be picked up and the Boolean function `Alertcheck()` initialized with the parameter received from SNORT. The `Alertcheck()` function verifies the authenticity of the attack by comparing it to the parameters in the linked list containing each host's information.

First, the SID function would be analyzed for the type of attack which was initiated, precisely what type of operating system was involved, the port which was attacked, and the service running on that port on the given IP address for that machine. The processing of finding a vulnerable signature ID match would take place using a 3D Linked List similar to the one that SNORT uses to find Rules and Options. The list headings would contain the IP numbers of the hosts on the network we are trying to protect.

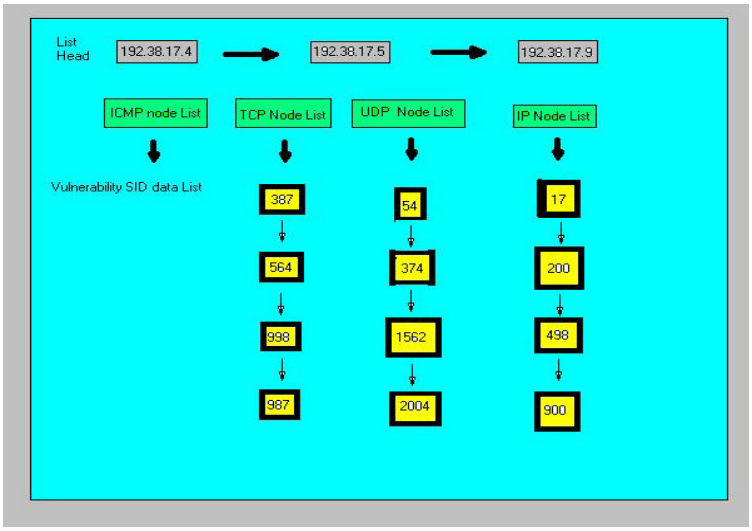

Figure 8.3 Vulnerability Linked List

For each of the IP number chains, there are separate linked lists, organized by the protocol that the possible attack was using. The four possible protocols implemented by our 'Vulnerability List' are the same as the SNORT rule list

1. UDP (UDP Protocol) —e.g., Domain Name Service Lookups
2. TCP (TCP Protocol)— e.g., FTP, HTTP, SMTP
3. IP (IP Protocol)— e.g., IGMP
4. ICMP (ICMP Protocol)— e.g., ping, traceroute, nslookup

Within each of these protocol linked lists are the SIDs to which each host listed is vulnerable. For instance, if an attack were made using SID 358 on host 192.34.23.2:

- The object would be received from the SNORT sensor.
- The `Alertcheck()` function would then be called with the object from the SNORT sensor with IP attack information and the number of the SID, in this case 358.
- The Header node search would stop at 192.34.23.2.
- The protocol list containing the TCP vulnerabilities list would then be searched to find the integer value 358. If found, then this message from the SNORT sensor will produce an alert, although more information and other checks will be made first. (Refer to Figure 8.3)

On initialization of our alert management program, the vulnerability list files would be read and population of the linked lists would take place. Once `Alertcheck()` has made a determination of whether the system is vulnerable, a selection statement would choose to send an alert to the AlertScreen or the information is logged using `StoreIt()`.

8.5.3 `StoreIt()`

`StoreIt()` simply reads the information from the object received in an unchanged form from `GetParam()` and logs the information in each respective field based on the Priority number in the object.

Table 8.1 Priority One Table

SID	TIME	PROTOCOL	PRIORITY	PORT	SOURCE IP	DESTINATION IP	MESSAGE
145	2003040304:04:02PM	tcp	1	21554	192.78.2.24	197.87.90.4	BACKDOOR GirlFriendaccess
213	2003040310:27:02PM	tcp	1	23	192.78.2.29	197.87.90.4	BACKDOOR MISC Linux rootkit attempt
312	2003040312:00:02PM	udp	1	123	192.78.2.29	197.87.90.11	EXPLOIT ntpdx overflow attempt
729	2003040312:17:02PM	tcp	1	25	192.78.2.29	197.87.90.4	VIRUS OUTBOUND .scr file attachment
900	2003040312:27:02PM	tcp	1	*	192.78.2.24	197.87.90.11	WEB-CGI webspirs.cgi directory traversal attempt
872	2003040312:29:02PM	tcp	1	*	192.78.2.29	197.87.90.17	WEB-CGI tcsh access

The Priority One table would include all the dangerous attacks which might cause real harm to the system.

Table 8.2: Information Gathering Attacks

SID	TIME	PROTOCOL	PRIORITY	PORT	SOURCE IP	DESTINATION IP	MESSAGE
624	2003040304:04:02PM	tcp	scan	*	192.78.2.24	197.87.90.4	SCAN SYN FIN
630	2003040312:00:02PM	tcp	scan	*	192.78.2.29	197.87.90.4	SCAN synscan portscan
1133	2003040312:00:32PM	tcp	scan	*	192.78.2.29	197.87.90.4	SCAN cybercop os probe
729	2003040312:01:02PM	icmp	scan	*	192.78.2.29	197.87.90.4	ICMP Broadscan Smurf Scanner
1918	2003040312:34:02PM	icmp	scan	*	192.78.2.24	197.87.90.4	SCAN SolarWinds IP scan attempt
2268	2003040312:50:12PM	smtp	scan	*	192.78.2.29	197.87.90.4	SMTP MAIL FROM sendmail prescan too long addresses overflow

The Information Gathering Attacks table would include any type of port scanning behavior, which is typical of hacker behavior.

Table 8.3: Low Priority Attacks

SID	TIME	PROTOCOL	PRIORITY	PORT	SOURCE IP	DESTINATION IP	MESSAGE
321	2003110 304:04:0 2PM	tcp	3	*	192.78.2.24	197.87.90.4	FINGER account enumeration attempt
401	2003110 304:07:0 2PM	icmp	2	*	192.78.2.29	197.87.90.4	ICMP Destination Unreachable
700	2003110 304:09:4 2PM	tcp	5	*	192.78.2.29	197.87.90.4	MS-SQL/SMB xp_updatecolvbm possible buffer overflow

The Low Priority Attacks table would include all the secondary attacks which might cause nuisances to the system. Possible root control or other extremely serious host side effects would not be included in this table.

All three tables are of the same format and each record would contain eight character Strings, named after each field in the corresponding SNORT sensor object. “Priority One” information is stored in the Priority One Table (Table 8.1). Port scans and other information are stored in the Information Gathering Attacks table, and all others fall into the default Low Priority Attack table. All fields are included in the tables and logged as such. If the SID number was found in the Vulnerability List, then `CreateAlert()` is called.

8.5.4 `Create Alert()`

In a situation where `Create Alert()` is called, then an attack has been made, SNORT has identified it, and RAMIT will have guaranteed that a vulnerability exists on that host. In other words, the network is clearly in danger and it is time to react. `Create Alert(),` taking the parameter obtained by `GetParam(),` would initiate SQL commands on

the table containing port scans. Typical hacker behavior, as observed over time, dictates that a port scan is standard operating procedure before making an attack.

Once the offender's IP number is in the object we are using, our database command will look up any previous port scan attempts by the same IP. Then a command would be issued to determine if the IP number of the attacker has made a port scan attempt on the host now under attack.

Afterwards, the Priority One table and the Low Priority Attacks table would be given SQL commands to search for the attacking IP number. This information would be stored in a new *ALERT* object and will be used when sent to the alert screen to notify the system administrator. Next, the victim IP number is searched for in the Priority One, and Low Priority tables. With that information in hand, RAMIT will be ready to send an Alert from Level 1 to Level 5.

8.5.4.1 RAMIT Level 1 Alert. Alert Status Level 1 is very similar to a standard SNORT alert except for the marked improvement in that RAMIT will have verified that an actual vulnerability exists on that host before issuing the alert. SNORT would simply match this packet contents with its rule base, and then based on its operation and no other knowledge, send an alert. RAMIT would receive this alert parameter and find that a vulnerability does indeed exist. Then RAMIT would search its database looking for other activity from this IP number. Level 1 cases will be created when no previous activity exists in the database and the search is negative.

Alert Status Level 1:

- NO SCANS by Attacking IP Number
- NO Attacks found by Attacking IP Number
- Vulnerability found to exist on host 196.38.17.3

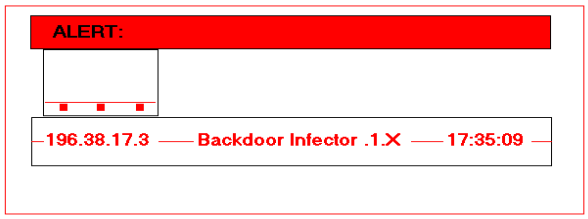

Figure 8.4 RAMIT Level 1 Alert

8.5.4.2 RAMIT Level 2 Alert. Alert Status Level 2 would come into play when an actual vulnerability exists on the host being attacked and port scanning activity has been found from the same IP address. Again, SNORT would simply match this packet contents with its rule base, and based on its operation and no other knowledge, send an alert. RAMIT would receive this alert parameter and find that a vulnerability does indeed exist. Then RAMIT would search its database looking for other activity from this IP number. Level 2 alerts will be created when no previous attacks have occurred from this IP; however, previous port scanning activity on other hosts exists in the database and the search is positive.

Alert Status Level 2:

- SCAN found for Attacking IP Number
- No SCAN found for host being attacked
- NO Attacks found by Attacking IP Number
- Vulnerability found to exist on host 196.38.17.3

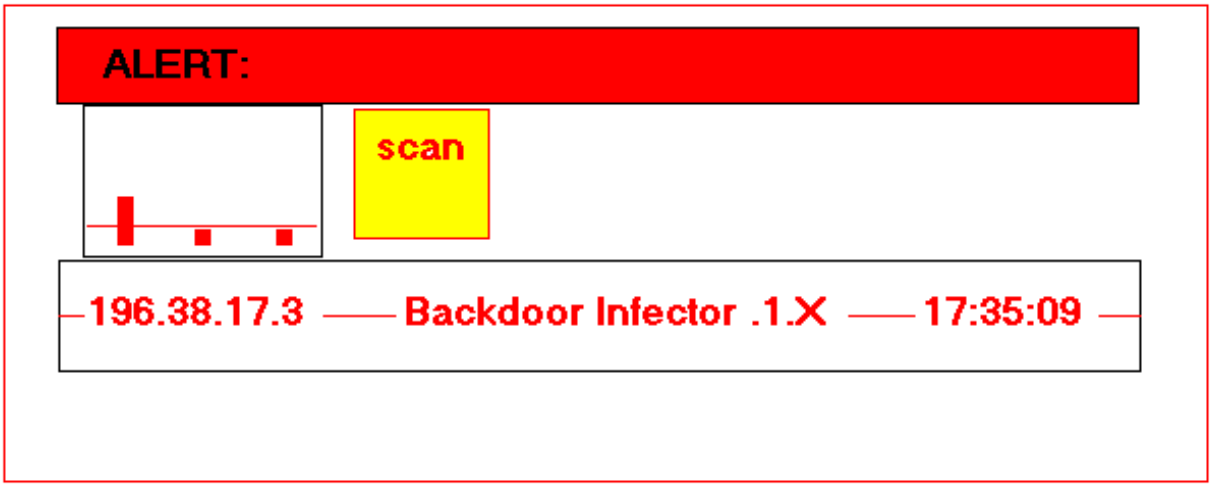

Figure 8.5 RAMIT Level 2 Alert

8.5.4.3 RAMIT Level 3 Alert. Alert Status Level 3 is called when an actual vulnerability exists on the host being attacked and port scanning activity has been found from the same IP

address on the Destination Host. Once again, SNORT simply matches the packet contents with its rule base, and sends an alert. RAMIT would receive this alert parameter and find that a vulnerability does indeed exist. Then RAMIT would search its database looking for other activity from this IP number. Level 3 cases are created when no previous attacks have occurred by this IP, but previous port scanning activity on this destination host exists in the database.

Alert Status Level 3:

- SCAN found for Attacking IP Number
- SCAN found for host being attacked by Attacking IP number
- NO Attacks found by Attacking IP Number
- Vulnerability found to exist on host 196.38.17.3

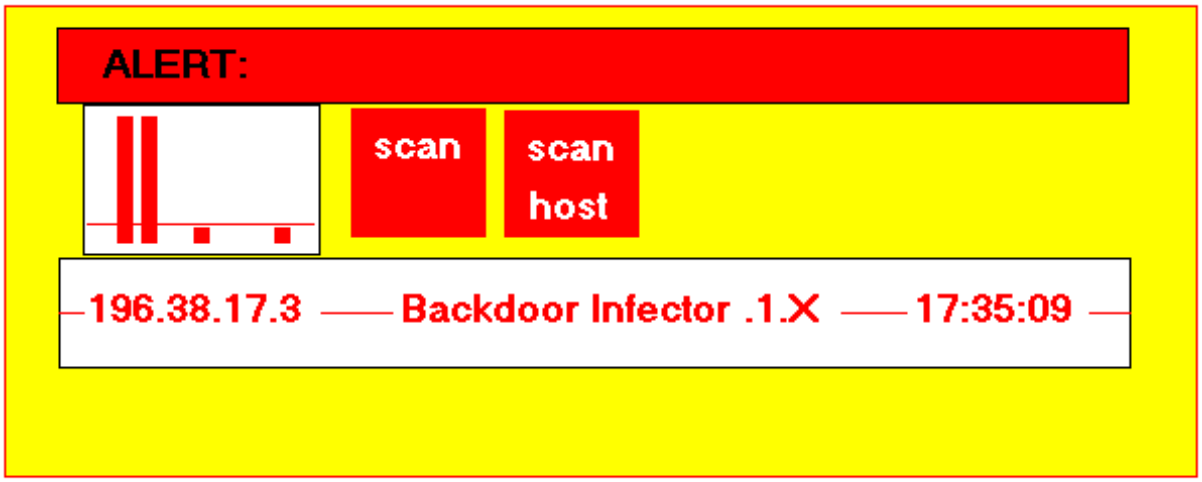

Figure 8.6 RAMIT Level 3 Alert

8.5.4.4 RAMIT Level 4 Alert. Alert Status Level 4 exists for a case in which an actual vulnerability exists on the host being attacked and port scanning activity has been found from the same IP address on the Destination Host and a low priority attack has been attempted by this intruder in the past. SNORT simply matches the packet contents with its rule base, and sends an alert. RAMIT would receive this alert parameter and find that a vulnerability does indeed exist.

Then RAMIT would search its database looking for other activity from this IP number. Level 4 cases are created when previous attacks have occurred by this IP, and previous port scanning activity on this destination host exists in the database.

Alert Status Level 4:

- SCAN found for Attacking IP Number
- SCAN found for host being attacked by Attacking IP number
- Attack found by Attacking IP Number (Low Priority- Default table)
- Vulnerability found to exist on host 196.38.17.3

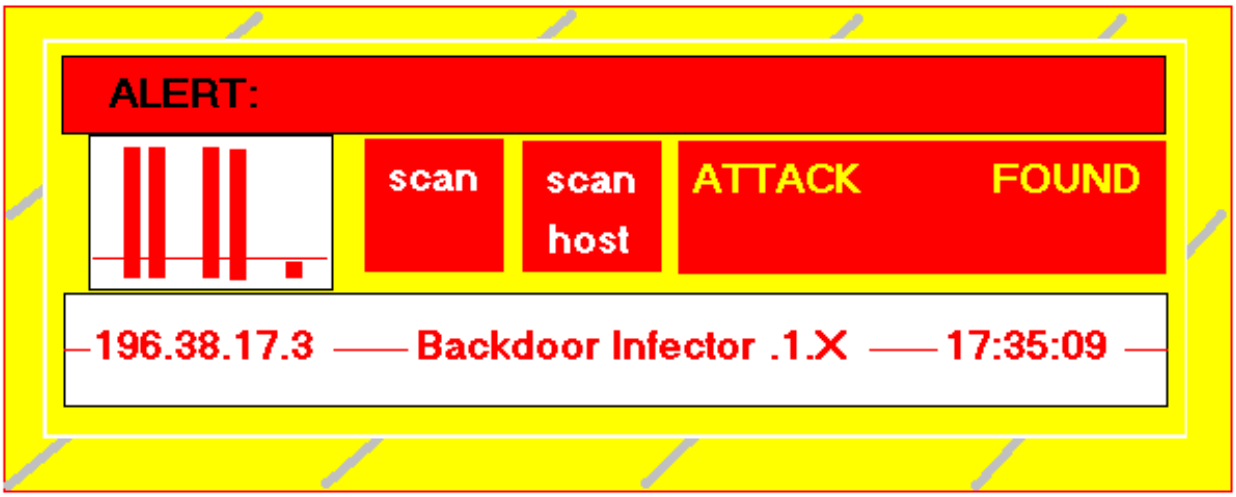

Figure 8.7 RAMIT Level 4 Alert

8.5.4.5 RAMIT Level 5 Alert. Alert Status Level 5 exists when an actual vulnerability exists on the host being attacked and port scanning activity has been detected from the same IP address on the Destination Host and a Priority One attack has been attempted by this intruder in the past. RAMIT would receive this alert parameter and find that a vulnerability does indeed exist and then search its database looking for other activity from this IP number. Level 5 cases are created when previous priority one attacks have occurred by this IP, and previous port

scanning activity on this destination host exists in the database. This is a worst case scenario and the alert reflects that.

Alert Status Level 5:

- SCAN found for Attacking IP Number
- SCAN found for host being attacked by Attacking IP number
- Attack found by Attacking IP Number (HIGH Priority- Priority I table)
- Vulnerability found to exist on host 196.38.17.3

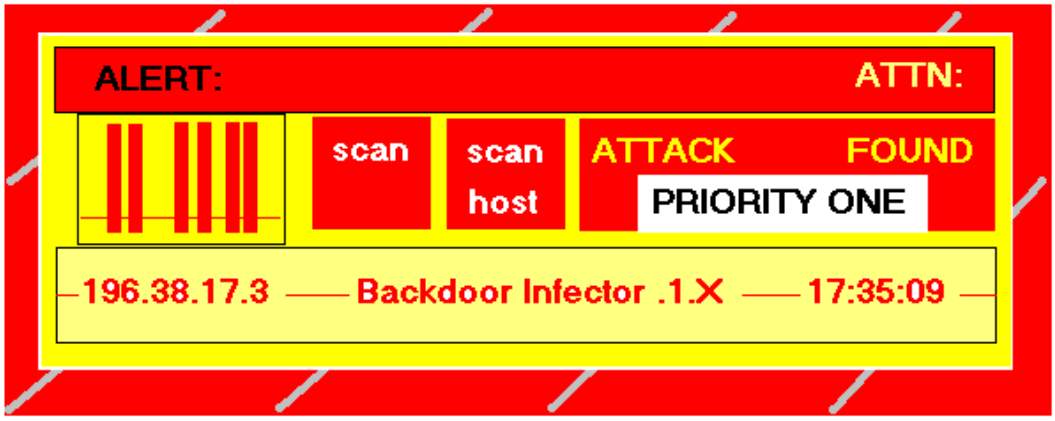

Figure 8.8 RAMIT Level 5 Alert

Once the data is sent, it would be formatted in an HTML browser using `Display()` which 'listens' for any new data to display, and then the information received in the SNORT sensor object is placed in the Tables using `StoreIt()`.

The improvements that RAMIT would offer over SNORT's basic services increase in each of these cases. As more and more pertinent information is found, the chances increase that an alert represents a dangerous attack. SNORT has no vulnerability information, let alone any intruder history. With RAMIT, each time the Alert Level rises, the likelihood increases that the attack is coming from an actual malicious hacker; concurrently, the likelihood increases that the system administrators, in real time, can respond effectively in order to defend the system.

CHAPTER NINE

SYSTEM EVALUATION

9.1 Overhead

It must be mentioned that since RAMIT would run on a separate server on the network, an additional computing machine would be required for implementation and operation. The RAMIT server would need both a large amount of RAM and voluminous drive space in order to handle the amount of data inside the database for both the intruders and the network awareness module list. Also, ultra high speed ethernet cards would be necessary for the amount of traffic received by the multiple SNORT sensors. Communication architecture inside the network would play an important role as well so that the traffic (SNORTsensor-RAMIT) would not unduly disrupt the rest of the network. RAMIT should most likely be part of a secured "switched" area in a low traffic area.

Another important feasibility problem is the lengthy and difficult nature of the database searches that must take place each time an alert is sent to the RAMIT server. It is imperative that proper indexing and views be used inside all of our database tables to permit the most rapid acquisition of data possible. In doing research in the IDS field, Mcaffee found that it was unproductive for an IDS to be able to find a dangerous packet and then attempt to adjust the firewall to shutdown the port for the attacking IP number. While this could be done systematically, it would be impossible to determine if this action truly circumvented an attack. Not only was this impossible for speed reasons, but it was also determined that if an attack was detected it could not be established if the attack was the opening attack made by the attacker or a second, third or later attack on the host. The problem here is that a host cannot be 'automatically protected' without some intervention by a human, if an alert is found by the administrator. In

other words, damage done to the host in a short amount of time could be used or accessed by the hacker which would have nothing to do with the actual attack that was detected. For our purposes this means that our proposition of a knowledge gathering device would give the administrator a much better tool than one than tried to intervene for the administrator. Also because of this, speed is now much less of an issue. A proper systems procedure, which assumes a violation on the host, could now methodically protect the host through human intervention, as long as the administrator is given a warning in a reasonable amount of time.

However, one of the benefits of RAMIT which counterbalances this new overhead is that the tool centralizes inputs from all SNORT sensors on a network under attack. In this way correlations between attacking IP numbers can improve the accuracy and probability of an actual attack. It is important to note that all SNORT logs remain local on the machine with the sensor and only alert possibilities are concerned with the centralized alert management tool.

At the bottom line, none of the ability of SNORT is compromised and it will continue logging packets and signatures from packets the way it always has. What is changing is that alerts in real-time are accessible on the centralized alert management machine. Because of this, data forensics will not change, and the system administrator, if needed, will be able to look at all the information that was present and logged.

9.2 Central Point of Failure

It is important to make mention of the central architecture of this system. Although a great asset, the central server that RAMIT exists on also is certainly a potential weakness. If by chance, an attacker were to gain control of the machine running RAMIT, the consequences would be catastrophic. Separate encrypted channels are a must, connecting only the RAMIT server system with the SNORT sensor nodes. All other ports should be shut down and the machine itself should sit behind a protected switch on its own subnet. Any known security problems with the operating system running RAMIT should be identified and patched immediately. Also, since the only outer point of contact should be a machine running a SNORT sensor, it is imperative that the administrator remain watchful in case the SNORT machine becomes compromised. A program such as Tripwire or another file analysis program should be monitoring changes to the RAMIT database. Also, anyone attempting ‘root’ control powers

would be a definite indication that a SNORT sensor machine had been hacked. With all ports closed, except necessary SNORT channels, and a vigilant program monitoring unauthorized behavior on the system, RAMIT could remain relatively secure.

9.3 Statistics of False Positives

In a recent paper dated February 2003 from the University of Tasmania, a statistical analysis was made concerning SNORT and False Positives.

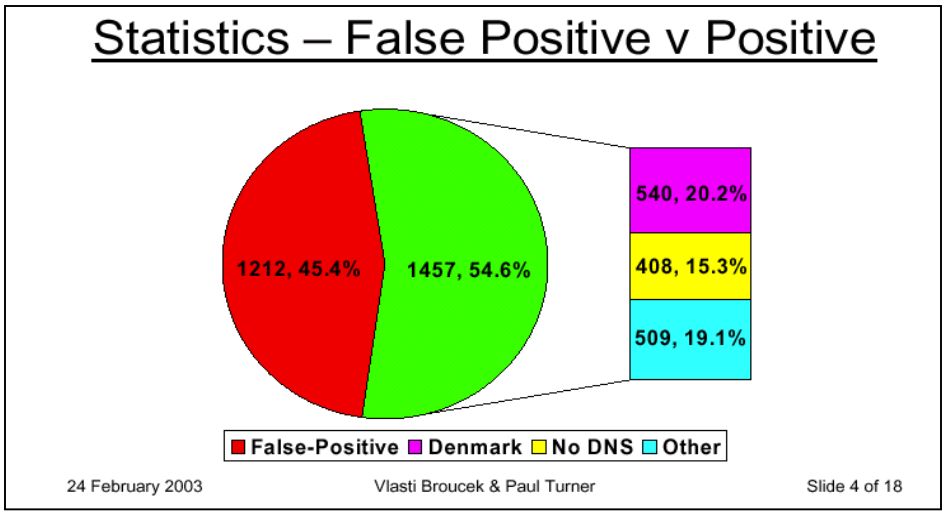

Figure 9.1 False Positive vs. Positive.
Source: http://brouk.psychol.utas.edu.au/files/TASIT2003.pdf

In this test, out of a total 2,669 attacks, exactly 1,212 were false positives—nearly half of all the alerts generated. Clearly there is a huge problem concerning these false alerts. Imagine the psychological effect this situation must have on the mind of a system administrator. Just like the villagers and the boy who cried wolf, with so many of the alerts turning out to be false, the system administrator may not even take the real alerts seriously. Using RAMIT to cross check and verify these alerts would allow the administrators to use better judgment in their responses

and reserve their energies for true threats to the network, thus increasing their efficacy and efficiency.

9.4 False Alerts

FTP attacks present a common, daily problem for system administrators. On a moderate size network the number of FTP attacks in one day could range from the hundreds to even the thousands. By using RAMIT and the proper placement of patches on machines, many of these worthless threats could be eliminated.

First, if FTP has been disabled or a patch is in place, then the FTP attacks would be ineffective. So instead of looking at hundreds of these kinds of alert messages:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP EXPLOIT format string";
flow:to_server,established; content: "SITE EXEC |25 30 32 30 64 7C 25 2E 66 25 2E 66 7C 0A|";
depth: 32; nocase; reference:cve,CVE-2000-0573; reference:bugtraq,1387;
reference:arachnids,453; classtype:attempted-user; sid:338; rev:5;)
```

Figure 9.2 FTP Attack Alert. Source: [HO02].

the system administrators would be freed to attend to more productive and pressing tasks. With RAMIT however, not only will useless or benign attacks be filtered, but the IP number the Attacker used is logged for future reference and will be searched out if another, possibly vulnerable host, is attacked. Remember that unless the attack represents an actual vulnerability, RAMIT will not alert the Administrator. By its operational logic, RAMIT cannot give a false positive. Although SNORT determines if an attack has occurred, RAMIT makes the determination if it is an actual threat or not.

9.5 Using Knowledge of Attacks

By examining an actual attempt on a network that was being monitored by SNORT, we will compare the RAMIT output to the default SNORT alert messaging [HO02]. In this example,

scans were implemented after the network firewall was pinged at 14:11. At 14:25, two network SNORT sensors picked up scans on TCP port 111. The following was logged by SNORT:

```
[**] TCP Traffic [**]
03/31-14:23:04.241672 0:50:BA:B2:CB:A3 -> 0:B0:D0:77:A5:57 type:0x800
len:0x4A
63.146.69.91:4245 -> x.x.x.x:111 TCP TTL:51 TOS:0x0 ID:57390 IpLen:20
DgmLen:60 DF
******S* Seq: 0xE44427A4  Ack: 0x0  Win: 0x7D78
TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 101462061 0 NOP WS: 0
```

Figure 9.3 Attack Log 1. Source: [HO02].

```
[**] TCP Traffic [**]
03/31-14:23:04.242193 0:B0:D0:77:A5:57 -> 0:50:BA:B2:CB:A3 type:0x800
len:0x4A
x.x.x.x:111 -> 63.146.69.91:4245 TCP TTL:64 TOS:0x0 ID:12278 IpLen:20
DgmLen:60 DF
***A**S* Seq: 0x78FEB002  Ack: 0xE44427A5  Win: 0x7D78  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 73973544 101462061 NOP
TCP Options => WS: 0
```

Figure 9.4 Attack Log 2. Source: [HO02].

```
[**] TCP Traffic [**]
03/31-14:23:04.400141 0:50:BA:B2:CB:A3 -> 0:B0:D0:77:A5:57 type:0x800
len:0x42
63.146.69.91:4245 -> x.x.x.x:111 TCP TTL:51 TOS:0x0 ID:57413 IpLen:20
DgmLen:52 DF
***A**** Seq: 0xE44427A5  Ack: 0x78FEBD03  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 101462078 73973544
```

Figure 9.5 Attack Log 3. Source: [HO02].

Finding that port 111 was open and vulnerable, a connection was made. Then the following scan was made.

```
[**] UDP Traffic [**]
03/31-14:23:04.410190 0:50:BA:B2:CB:A3 -> 0:B0:D0:77:A5:57 type:0x800
len:0x62
63.146.69.91:1023 -> x.x.x.x:111 UDP TTL:51 TOS:0x0 ID:57416 IpLen:20
DgmLen:84
Len: 64
13 F6 70 D6 00 00 00 00 00 00 00 02 00 01 86 A0  ..p.............
00 00 00 02 00 00 00 03 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 01 86 B8 00 00 00 01  ................
00 00 00 11 00 00 00 00                          ........
```

Figure 9.6 Attack Log 4. Source: [HO02].

```
[**] UDP Traffic [**]
03/31-14:23:04.410778 0:B0:D0:77:A5:57 -> 0:50:BA:B2:CB:A3 type:0x800
len:0x46
x.x.x.x:111 -> 63.146.69.91:1023 UDP TTL:64 TOS:0x0 ID:12279 IpLen:20
DgmLen:56
Len: 36
13 F6 70 D6 00 00 00 01 00 00 00 00 00 00 00 00  ..p.............
00 00 00 00 00 00 00 00 00 00 03 B3              ............
```

Figure 9.7 Attack Log 5. Source: [HO02].

which shows the attacker trying to determine what port `statd` was running on. The intruder found that host ran *rpc* and also `statd` on udp port 947. He then checked if the version of `statd` running was vulnerable [HO02]. Malicious shell code was then sent to make `statd` produce a root prompt. Again this was discovered by SNORT and represented the first time a vulnerability was encountered instead of simply information gathering.

RAMIT would handle a situation like this a bit differently.

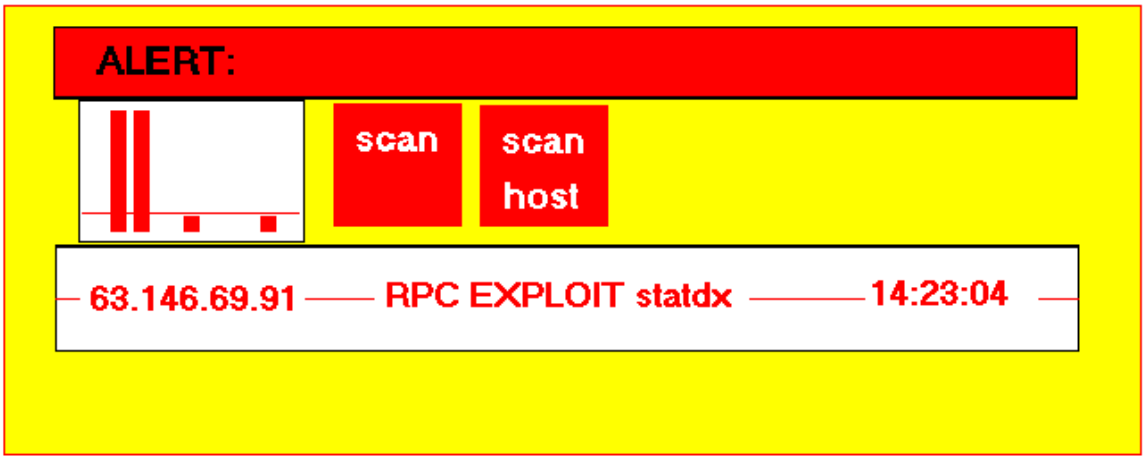

Figure 9.8 `statd` Alert Level 3

First, although the port scanning would be watched and logged no alert would be sent until the *statd* spoof. Instead of five attack logs to occupy the system administrator, just one would be necessary to convey the pertinent information for the situation at hand. Along with the *statd* spoof alert, the port scans preceding it would be logged as well, as well as whether this particular hacker had attempted anything on other network hosts and the nature of the methods he used. In this case, the attacker scanned the host but at no other time did he, based on IP, attempt an attack. This time the attacker rated a Level 3, the attack was alerted in real time and if the attacker tried again anywhere on this monitored network, he would rate a Level 5 alert.

9.6 Unique IP Numbers

It is important to mention that the primary benefit of this system is representing the vulnerabilities of each host so that an improper false alert is no longer an issue. The added benefit of this system is 'knowledge history' context of attackers. It is obvious that a wily attacker may choose to use different machines to make attacks, trying to avoid detection. Using DHCP or a similar tool, it would be evident that the IP number would change, even for the same machine. Also, logging in remotely into various networks using this technology might at first seem to evade and render useless the 'added' features which RAMIT provides. However, by using RARP or other methods the MAC address of the Ethernet cards could be easily found, and then DHCP or anything else, which might hinder proper identification of the attacker, would be instantly transparent.

Honeypots have made researchers aware that hackers typically, like most human beings, favor the 'least effort' method to gain access to host machines. In the unlikely event that someone is willing to go to the greatest lengths to hide their steps, RAMIT could be modified over time to subdivide the attacking IP numbers into "equivalence classes" based on type of network. Then, instead of RAMIT trying to determine attacker IPs, RAMIT would simply note the network as being at fault and then look at the future attacks for IP numbers from that network number.

9.7 Summation

There is no “magic bullet” solution to the problem of network intrusion. As argued throughout this thesis, even the most successful commercial products have significant weaknesses. That said, there is still a need to always strive for improvements upon the defenses currently available. As proposed here, RAMIT represents a significant, though not final, step forward in that quest.

CHAPTER TEN

CONCLUSION

Any network that presents some degree of a challenge to hack will, by its very existence, invite attack. As necessity is the mother of invention, programmers have developed and refined various security measures including such network intrusion detection systems as SNORT. However, despite SNORT's relative success, there is room for improvement.

As previously stated, SNORT, on its own, cannot identify a previous intrusion or intruder attempt relative to the current time because the packet involved is only important in the current scan and not in a historical context. RAMIT's design improves on this concept greatly. By maintaining a database of both intruders and vulnerabilities, not only would a previous hacking attempt be known, it would be cataloged and referenced each time an attack occurs. Another benefit exists in that not only could an intruder be identified, but how potentially dangerous he is could be determined by analyzing the vulnerability of the attack.

RAMIT would maintain a network awareness by using its vulnerability linked list and could make decisions that SNORT would be incapable of. Far more than just a packet sniffer and string matcher, RAMIT would use a knowledge base to make intelligent decisions on whether an alert is a real threat. In using this knowledge base, hosts could be analyzed for their susceptibility to attacks. Definite patterns of hacking attempts based on analysis of snort logs in correlation with RAMIT alerts would aid the system administrator greatly in securing his network system.

Computer science, more than any other field, strives continually to improve and perfect its systems on a daily, even hourly, basis. Some of the most effective advances have come not from reinventing the wheel, so to speak, but in doing as Sir Isaac Newton did when he modestly claimed to have but stood on the shoulders of giants who had gone before.

While a network intrusion detection system might not be as noteworthy a development as the internal hard drive, it plays a more vital daily role than many users, particularly corporate and

university-based, might ever know. Any improvement to such a system will be of even greater benefit to those computer professionals who could, without those excess responsibilities, be of even greater service to their community, thus continuing to pass on the benefit in a trickle down fashion.

As a case in point, if a tool such as RAMIT were to be incorporated as a part of the existing battery of security devices, the benefits both to users and administrators would be clear and discrete. Suspicious behaviors would be more thoroughly investigated before involving human resources. Any such attack behaviors would be logged in the RAMIT database and used as part of the decision making for future alerts. Events would be generated in real time, and multiple sensor inputs are input and analyzed centrally. False negatives should be dramatically reduced and general "noise" held to a minimum. Anything crossing the RAMIT rule base should be serious and would call for immediate action.

Another clear benefit is the construction of a “multiple offender list”. Similar in essence to the sexual offender registry, once suspicious behavior has been identified, the entity employing that IP address is marked as suspect and closely tracked. If the intruder probes another part of the network and is detected by SNORT, that invasion is matched with the prior offense and the appropriate response is triggered by RAMIT. This presents a value-judgment tool that was not available before. At the bottom line, any reduction in attacks would result in a more profitable and peaceful working experience.

And yet, as with all developments, there could be room for improvement with RAMIT. A host vulnerability tool which identifies possible intrusion windows and correlates these with the database of SNORT rules could be invaluable. This information, needed for the RAMIT vulnerability list, would be that much easier to populate. The proverbial Holy Grail of system administrators would be a method to anticipate a denial of service attack, but that remains as elusive as the hacker psyche.

Common wisdom states that for every gain there must be a loss, for every step forward, one back, no pain no gain, ad infinitum. The beauty of an integrated alert management tool such as RAMIT is that these pessimistic adages simply don’t apply. RAMIT provides a number of distinct improvements to an already proven network detection tool and at such little cost that it would be practically unthinkable for conscientious system administrators not to implement it, both for their users and for themselves.

BIBLIOGRAPHY

[BAC99] Bace, Rebecca. "An Introduction to Intrusion Detection and Assessment: for System and Network Security Management," ICSA White Paper, 1998.

[CBFP03] Caswell, Brian; Beale, Jay; Foster, James C.; Posluns, Jeffrey. "Snort 2.0 – Intrusion Detection" 2003 Syngress Publishing.

[CNN00] http://www.cnn.com/2000/TECH/computing/09/06/fear.trinity.idg/

[COD00] Coddington, Dale "Snort Installation and Basic Usage - Part Two" 2000 http://www.securityfocus.com/infocus/1422

[FR00] Freeman-Hargis, James "Introduction to Rule-Based Systems*"* http://ai-depot.com/Tutorial/RuleBased.html

[FRED1569] Frederick, Karen Kent *Infocus* "Network Intrusion Detection Signatures, Part 5" http://www.securityfocus.com/infocus/1569

[HG03] Hulme, George V. "Gartner: Intrusion Detection on the Way Out."

[HO02] Holcroft, Stephen. "Incident Analysis of a Compromised RedHat Linux 6.2 Honeypot" April 2000 http://mrcorp.infosecwriters.com/contribs/Incident_Analysis_6_2%20Honeypot.htm

[HP03] Hollows, Phil. "IDS is Dead—Long Live IDS?" http://itmanagement.earthweb.com/columns/article.php/2228631

[HOL02] Holstein, Michael. **"**How does Fragroute evade NIDS detection?" 2002 http://www.sans.org/resources/idfaq/fragroute.php

[INMC01] Innella, Paul; Mcmillan Oba. "An Introduction to Intrusion Detection Systems" 2001 http://www.securityfocus.com/infocus/1520

Lemos, Robert. New tool camouflages hacker programs. ZdNet Australia. 22 April 2002. http://www.zdnet.com.au/newstech/security/story/0,2000024985,20264745,00.htm

Mitre. Common Vulnerabilities and Exposures. 27 August 1999. http://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0082

[NFR2003] http://www.nfr.com/solutions/technical.php

[OL00] Olsen, Anne. “Artificial Intelligence” 2000 http://www.slais.ubc.ca/courses/libr500/2000-2001-wt1/www/a_olsen/

[PEICHU04] Peikari, Cyrus. Chuvakin, Anton; “Security Warrior” 2004 O’Reilly & Associates

[POW99] Power, Richard. "1999 CSI/FBI Computer Crime and Security Survey," Computer Security Journal, Volume XV, Number 2, 1999, pp. 32.

[PTA02] Ptacek, Thomas H. “Insertion, Evasion, and Denial of service: Eluding Network Intrusion Detection” 2002 Secure Networks http://WindowSecurity.com

Ptacek, Thomas & Newsham, Timothy. “Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection”. Secure Networks, January 1998. http://www.insecure.org/stf/secnet_ids/secnet_ids.html

[RM03] Ranum, Marcus J. “False Positives: A User’s Guide to Making Sense of IDS Alarms.” ICSA Labs IDSC. http://www.icsalabs.com/html/communities/ids/whitepaper/FalsePositives.pdf

Roesch, Marty. News. 7 May 2002. www.snort.org/index.html

[SANS01] SANS Institute staff, "Intrusion Detection and Vulnerability Testing Tools: What Works?" 101 Security Solutions E-Alert Newsletters, 2001.

Song, Dug. “Fragroute(8)” http://www.monkey.org/~dugsong/fragroute/fragroute.8.txt

[SNORT] www.snort.org

[TIMBER] http://www.timberlinetechnologies.com/products/intrusiondtct.html

Timm, Kevin. IDS Evasion Techniques and Tactics. SecurityFocus (Infocus). 7 May, 2002 http://online.securityfocus.com/infocus/1577

[US01] Unix Security. “The Effect of Crying Wolf.” http://www.itworld.com/nl/unix_sec/10252001

[WEBO] http://www.webopedia.com/TERM/T/Trojan_horse.html

Yee Andre, “Making false positives go away” NFR Security 22 January, 2004 http://www.computerworld.com/securitytopics/security/story/0,10801,89122,00.html

BIOGRAPHICAL SKETCH

John Blackwell received a Bachelor's of Science degree from the University of Illinois Urbana-Champaign in 1990. He has worked at various positions as an engineer, dishwasher, headhunter, patrol officer, counselor for the mentally ill, MIS manager for a direct mail company, and as a college instructor in computer science. At the time of this paper, he is working as a senior Java developer and helping to bring the State of Florida, kicking and screaming, into the 21st century. He was lucky enough to marry the girl of his dreams and really has no idea what the Good Lord has for him next, although it is sure to be an adventure!