Spring 5-31-2006

Classification, testing and optimization of intrusion detection systems

Javier Leon

New Jersey Institute of Technology

ABSTRACT

CLASSIFICATION, TESTING AND OPTIMIZATION OF INTRUSION DETECTION SYSTEMS

by
Javier Leon

Modern network security products vary greatly in their underlying technology and architecture. Since the introduction of intrusion detection decades ago, intrusion detection technologies have continued to evolve rapidly. This rapid change has led to the introduction of a wealth of security devices, technologies and algorithms that perform functions originally associated with intrusion detection systems.

This thesis offers an analysis of intrusion detection technologies, proposing a new classification system for intrusion detection systems. Working closely with the development of a new intrusion detection product, this thesis introduces a method of testing related technologies in a production environment by outlining and executing a series of denial of service and scan and probe attacks. Based on the findings of these experiments, a series of enhancements to the core intrusion detection product is introduced to improve its capabilities and adapt to modern needs of security products.

CLASSIFICATION, TESTING AND OPTIMIZATION OF INTRUSION DETECTION SYSTEMS

by
Javier Leon

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Engineering

Department of Electrical and Computer Engineering

May 2006

APPROVAL PAGE

CLASSIFICATION, TESTING AND OPTIMIZATION OF INTRUSION DETECTION SYSTEMS

Javier Leon

Dr. Constantine Manikopoulos, Thesis Advisor Date
Associate Professor of Electrical and Computer Engineering, NJIT

Dr. Robert Statica, Committee Member Date
Information Technology Program Administrator, NJIT

Dr. Jie Hu, Committee Member Date
Assistant Professor of Electrical and Computer Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Javier Leon

Degree: Master of Science

Date: April 2006

Undergraduate and Graduate Education:

- Master of Science in Computer Engineering,
  New Jersey Institute of Technology, Newark, N.J., 2006

- Bachelor of Science in Computer Engineering,
  New Jersey Institute of Technology, Newark, N.J., 1999

Major: Computer Engineering

Presentations and Publications:

Leon, Javier Simms, Dennis and Redmond, Michael,
"A Strategy for Network Convergence,"
The Sixth International Symposium on Personal, Indoor and Mobile Radio
Educause, Sacramento, CA, October 2003.

To my beloved family, without which I would not have the courage and confidence to succeed.

ACKNOWLEDGMENT

I would like to express my deepest appreciation to Dr. Constantine Manikopoulos, who acted as my research supervisor, providing valuable and countless resources, insight, and intuition. I appreciate his efforts towards a deeper understanding of the technology behind network security. I would also like to extend my appreciation for the efforts of my committee members Dr. Robert Statica and Dr. Jie Hu.

Many of my fellow graduate students in the Computer Engineering program are deserving of recognition for their support. Witnessing the efforts of doctoral students under the leadership of Dr. Manikopoulos has provided me with a deeper appreciation of the benefits of graduate and doctoral research and the capabilities of motivated individuals with genuine leadership.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

CHAPTER 1

INTRODUCTION

1.1 Intrusion Detection

The birth of Intrusion Detection is most often credited to James Anderson's two published reports on computer security [1] [2]. In his 1972 paper, "Computer Security Technology Planning Study," Anderson identifies the need to "determine what constitutes an appropriate defense against malicious attack, and then develop hardware and software with the defensive mechanism(s) built in." In 1980 his paper suggested "an outline of a system design and the basis for providing statistical detection of abnormal use." Since these early beginnings intrusion detection technologies have quickly evolved. While computer security technologies have experienced tremendous change in the past two decades, many of the issues and solutions identified two decades ago are still true today.

Intrusion detection is defined by SANS [4] as "the art of detecting inappropriate, incorrect, or anomalous activity." Because of the complexity behind IDS technologies, opinions, definitions, and practices continue to differ significantly throughout the security industry. While products and technologies offering protection against security threats have continued to change, the number of reported vulnerabilities and incidents has continued to grow.

Between the years 2000 and 2003, the CERT Coordination center [3] has published an increase of over 340% in reported vulnerabilities.630% in reported security incidents. Between 1988 and 2002, the increase of reported vulnerabilities and incidents is over 1500% and 2100% respectively.

1.2 Intrusion Detection Mechanisms

Although changes to Intrusion Detection products and technology have led to a variety of different categories of products, the underlying technologies behind today's modern products are essentially the same. In this thesis, I propose the following definition of Network Intrusion Detection: A set of technologies designed to identify and report on indications of network policy violations.

Today's network security products tout the ability to predict, detect, eliminate or prevent network security violations. This can only be achieved with the assistance of intrusion detection mechanisms offering the ability to detect indications of policy violations. The Common Intrusion Detection Framework Architecture (CIDF) [5] defines the components of an Intrusion Detection System Architecture as Event generators ("E-boxes"), Event analyzers ("A-boxes"), Event databases ("D-boxes") and Response units ("R-boxes"). The Intrusion Detection mechanisms presented in this section would act as A-boxes in a complete IDS system. Analyzers receive information, performs analysis and return resultant information.

In the following analysis of intrusion detection mechanisms, various technologies employed by Intrusion Detection devices to detect violations are analyzed and a new system of categorizing these technologies is proposed.

Intrusion Detection Mechanisms can be separated into two main categories. Detection mechanisms will function by analyzing either the content of data being analyzed or the statistics generated by the analysis of the data. Content based mechanisms can be further categorized by the type of content analysis performed.

- Statistic Based
- Content Based
    - Signature Detection
    - Protocol Anomaly
    - Behavior

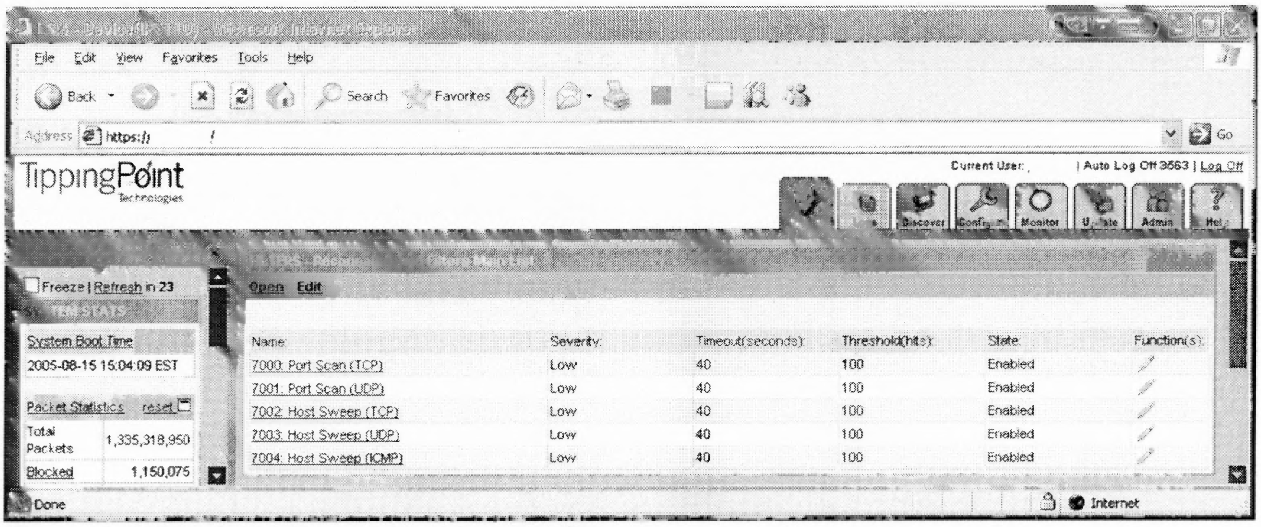
1.2.1 Statistic Based Mechanisms

Statistic based intrusion detection mechanisms use statistics generated from the analysis of observed data rather than the content of the data itself. Statistic based mechanisms act as anomaly detection tools by identifying anomalous behavior from gathered statistics.

Analyzed statistics range from basic packet counts to complex protocol analysis. Tipping Point's [6] Port Scan/Host Sweep reconnaissance filter detection analyzes five categories of statistics: (1) Port Scan TCP (2) Port Scan UDP (3) Host Sweep TCP (4) Host Sweep UDP (5) Host Sweep ICMP. Each of these filters is accompanied by a timeout and threshold value to determine the number of hits necessary in a given period of time that would indicate a port scan or host sweep. In contrast, RAP and HIDE algorithms [7] function by comparing from an exhaustive set of statistics to determine whether analyzed traffic matches the statistics of known attacks. In analyzing TCP traffic as many as six IP features and 41 TCP features can be analyzed.

Statistic based mechanisms are fundamentally anomaly based in the sense that it can only detect an event based on the comparison of current statistics to a baseline. The purpose of the baseline is to identify the statistical characteristics of what can be considered "normal" behavior. The baseline set of statistics to which data is compared can range from a set of basic data manually supplied by an administrator to the results of complex learning algorithms capable of determining baseline characteristics from complex analysis of live traffic.

Simple statistical analysis devices such as Tipping Point's Unity-400 [6] is made up of a set of total packet counts over a period of time categorized only by protocol type as seen in Figure 1.1. An anomaly is detected by comparing the defined value to that of the current packet count. RAP [7], on the other hand employs a complex analysis of a large set of data through a neural network in order to derive the resulting distribution statistics. These values will then be compared to the results of a complex learning algorithm that generates the "normal" distribution.



System Boot Time	
2005-08-15 15:04:09 EST	
Packet Statistics	reset
Total Packets	1,335,318,950
Blocked	1,150,075

Name:	Severity:	Timeout(seconds):	Threshold(hits):	State:	Function(s):
7000: Port Scan (TCP)	Low	40	100	Enabled	
7001: Port Scan (UDP)	Low	40	100	Enabled	
7002: Host Sweep (TCP)	Low	40	100	Enabled	
7003: Host Sweep (UDP)	Low	40	100	Enabled	
7004: Host Sweep (ICMP)	Low	40	100	Enabled	

Figure 1.1 Tipping point reconnaissance filters.
Source: [6]

1.2.2 Content Based Mechanisms

Content Based Intrusion Detection mechanisms are able to identify a security event based on the content of observed data. Such content can include fields within a single packet, packet payload, data transmitted throughout a communication session or any form of supported communication. Content based mechanisms can be classified into one of three types.

1.2.2.1 Signature Based Content Mechanisms. A fundamental method of detecting a security policy violation is through the use of signatures. A signature is a string of data, whose presence in network traffic indicates an intrusion detection event.

A challenge in the use of signature detection is the "zero day" problem. Commonly used to refer to the exploit of unremedied and often unidentified threats, the use of a signature database to store instances of such exploits provides no protection against those that have not yet been identified.

Examples of commercial Intrusion Detection Systems that are primarily signature based include ISS Realsecure [8], Symantec's Intruder Alert/Net Prowler [9]. Open source solution Snort [10] is an example of an IDS that is primarily signature based.

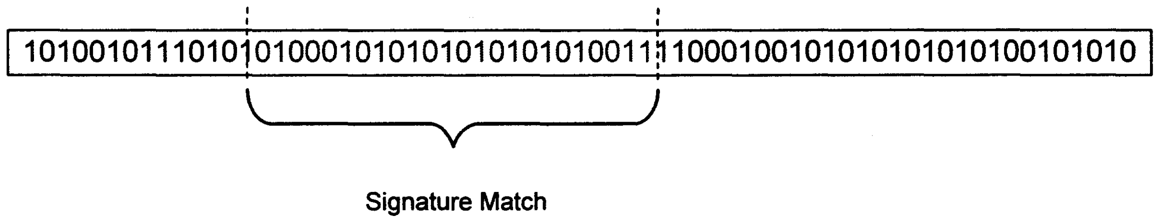


Figure 1.2 Signature detection mechanisms.

Signature based intrusion detection mechanisms are relatively simple mechanisms in that they rely on matching signatures to network traffic. Only direct matches of the identified string(s) will result in a related event. In order to effectively compare a signature to detected traffic, a system must be capable of accurately reconstructing and normalizing communication flows regardless of fragmentation, retransmission or similar methods capable of bypassing early intrusion detection mechanisms.

Effectiveness of signature based intrusion detection mechanisms can be enhanced through more effective handling of the comparison mechanisms or the creation of more accurate signatures. Many commercial products such as Top Layers IPS 550 [11], Tipping Point 2400 [12], and Juniper's ISG [13] rely on ASIC driven hardware and programmable ROMs to allow high speed comparisons at multi-gigabit speeds.

Snort [10] signatures consist of single statement rules that define the string to be matched against. The following example is a typical rule statement in Snort.

```
alert tcp any any -> any 139 (content:"|5c 00|P|00|I|00|P|00|E|00 5c|";)
```

Signature based intrusion detection algorithms are challenged by a rapidly increasing number of signatures.

Rule clustering technologies minimize the number of comparisons necessary for event detection. Similar methods of optimizing the comparison process include snort's parallelization method and ASIC based parallelization. The use of decision trees such as [14] assist in increasing the performance of signature based IDS systems by clustering rules into feature based sets that reduce the number of direct comparisons made by related algorithms.

Although signature detection has its inherent weaknesses, its value in an intrusion detection mechanism is extremely high. A well crafted signature will very effectively identify undesired traffic. Signature detection continues to play a critical role in most intrusion detection mechanisms

1.2.2.2 Anomaly Based Content Mechanisms. A protocol anomaly includes any traffic that violates an internet RFC. This category differs from signature detection in that the identification of such an event is triggered by a violation of a set of rules rather than the appearance of a specific bit pattern in a packet payload or data stream.

The behavior of specific network operating systems to normal network traffic as defined in internet RFCs can vary and has been the source of both fingerprinting mechanisms and network exploits. As a result, most intrusion detection systems are able to use deviations from defined standards as an indication of malicious activity. As with most intrusion mechanisms, protocol anomaly rules must be applied carefully in order to avoid false alarms that may arise from the variations in protocol stack of various operating systems.

Tipping Point's UnityOne [6] uses traffic Normalization filters to detect many of these violations. These include:

- IP Header Incomplete
- IP Fragment Invalid
- IP Fragment Out of Range
- IP Duplicate Fragment
- IP Length Invalid
- IP Fragment Total Length Mismatch
- IP Fragment Overlap
- IP Fragment Bad MF Bits
- TCP Segment Overlap With Different Data
- TCP Header Length Invalid
- TCP Flags Invalid
- TCP Header Incomplete

- TCP Length Invalid
- TCP Reserved Flags Invalid
- ICMP Header Incomplete
- ICMP Length Invalid
- UDP Header Incomplete
- UDP Length Invalid
- Ethernet Header Incomplete
- ARP Address Invalid
- ARP Header Incomplete
- ARP Length Invalid
- Unknown Traffic Normalization

1.2.2.3 Behavior Based Content Mechanisms. Behavioral anomaly mechanisms identify security policy violations by identifying behavior of communication session(s) that violates a set of behavioral guidelines. While the rules of internet protocols can be easily identified, web applications vary from site to site and have become a common target in many attacks. Traditional intrusion detection mechanisms were not well suited to identify violations of traffic policies defined by the existence of custom web based applications.

Behavioral anomaly mechanisms, when properly defined, are capable of identifying attacks that would otherwise be undetectable by other mechanisms by adapting to specific environments.

STATL [15] models penetrations as a series of state changes that lead from an initial secure state to a target compromised state. Rules are identified by a relatively complex language to define behavior as a set of well defined transition states. The following example demonstrates a scenario in STATL language that would detect half open TCP connections.

```
use netstat;
scenario halfopentcp(int timeout)
{
IPAddress victim_addr;
Port victim_port;
IPAddress attacker_addr;
Port attacker_port;
timer t0;
initial state s0 {}
transition SYN (s0 -> s1)
nonconsuming
{
[IP ip [TCP tcp]] :
(tcp.tcp_header.flags & TH_SYN) &&
!(tcp.tcp_header.flags & TH_ACK)
{
victim_addr=ip.header.dst;
victim_port=tcp.header.dst;
attacker_addr=ip.header.src;
attacker_port=tcp.header.src;
}
}
```

Bro, a network based IDS [16] uses a custom Bro scripting language to define sets of rules that would indicate an intrusion event. Both pre-written and custom scripts can be combined with a number of analyzers to create a set of "rules" that define normal and anomalous behavior in a network environment.

Reliable Software Technologies [17] combines the use of three machine learning algorithms (sting transducer, state tester and Elman Recurrent Neural Network) to analyze the use of common UNIX command line tools in order to detect attempts to misuse the command line tools.

"Although behavior-blocking is more effective against unknown viruses, it can still be fooled by carefully designed viruses that propagate slowly, or replicate after a period. For instance, if a system is set to block the behavior that an email attachment should not cause generation of more than k other email messages, a virus that generates only k-1 copies will go undetected. Similarly, an email attachment that causes time-delayed propagation may also go undetected. More generally, a virus can employ a combination of low propagation factor, high incubation period, and randomization to evade behavior-blocking approaches." [18]

In reaction to common workarounds for malicious software, behavioral based systems such as Stony Brooks' email propagation detection system [19] utilizes a combination of specification-based anomaly detection and statistical machine-learning mechanisms to more effectively detect email propagating viruses.

Academic designs like STAT [20] and Bro [21] are examples of systems capable of performing detection based on analysis of communication states. In the model presented, state change or state transition analysis techniques would be categorized as behavioral detection.

1.3 Conex Intrusion Detection System

In May 2004, Zheng Zhang's dissertation entitled "Statistical Anomaly Denial of Service and Reconnaissance Intrusion Detection" [22] led to a proof of concept design of the (RIDS) Reconnaissance Intrusion Detection System and (HIDE) Hierarchical Intrusion Detection Engine. The ongoing development of these systems would be driven in a large part by the results discovered in onsite testing.

HIDE is a packet-oriented intrusion detection mechanism that would use statistics discovered during sequential observation-windows to create a probability density function (PDF) for each of a number of monitored parameters. These PDF's would be compared to reference PDFs of normal behavior to create a similarity metric that would be compared to statistics of anomalous traffic through a neural network classifier. When tested in simulated and testbed environments, false positive rates were found to be as low as 0.090%.

RIDS, a session oriented tool, compares discovered session statistics to results of training algorithms to discover even stealthy network reconnaissance attacks. The RIDS mechanism consists of two primary components; RAP and RAC.

RAP is a session-oriented module capable of detecting stealthy scanning and probing attacks, while the RAC is an alert-correlation module that fuses the RAP alerts into attack scenarios and discovers the distributed stealthy attack scenarios. Misclassifications of attacks were recorded to be under .1% in simulated environment testing.

CHAPTER 2

CONEX IDS TESTING

2.1 Testing Environment

Testing of the IDS was to take place in a college campus networking environment. The environment consisted of a typical Cisco enabled internet and demilitarized zone configuration shown below. A fractional DS3 with a total of 20Mbps of Internet traffic is terminated in a Cisco 7206 router. Connected to the college's only Internet Service Provider, all internet traffic packets pass through this link, crossing a pair of Cisco PIX 525 firewalls along their path. The PIX firewall performs NAT and PAT functions on each packet in addition to comparing them against access control lists and performing basic DOS attack detection and mitigation. Connected to the PIX firewalls are the internal, external, and DMZ interfaces. Each of these interfaces is configured with a separate set of access control lists and network addresses.

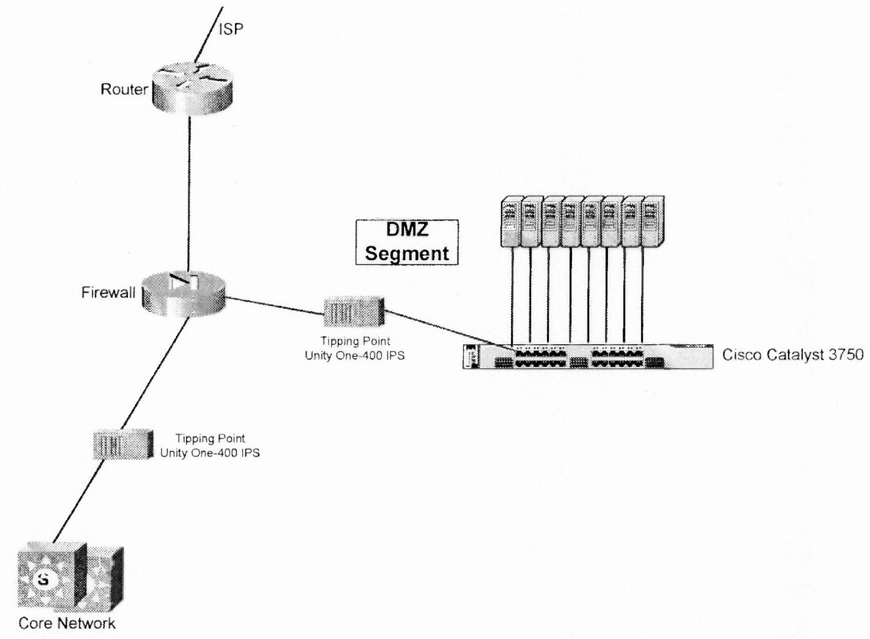

ISP
Router
DMZ Segment
Firewall
Tipping Point Unity One-400 IPS
Cisco Catalyst 3750
Tipping Point Unity One-400 IPS
Core Network

Figure 2.1 Existing test environment.

The DMZ and internal network segments are monitored by the college's UnityOne-400 Intrusion Prevention appliance. As an inline device, all inbound and outbound traffic traverses the security appliance and is protected by the configured rule set of the UnityOne appliance. The UnityOne-400 security appliance specifications are listed below [24].

- Inline device
- 400 Mbps total throughput capability
- 4 10/100Mbps interfaces
- Specialized Hardware Custom ASICs
- Scalable Parallel Processing
- Full Flow State Tracking
    - IP Fragment Reassembly
    - TCP Handshake
    - TCP Flow Reassembly
- Support for multiple Filtering Methods:
    - Application Anomaly
    - Protocol Anomaly
    - Traffic Anomaly
    - Signature
    - Full Regular Expressions
- <215 µsec Latency
- Capable of monitoring 2million simultaneous sessions
- Capable of tracking the creation of 250,000 sessions/second

Relevant UnityOne filters are organized into the following categories

- Application Protection – Protects from Worms, Viruses, Trojans, Internal Attacks, Unauthorized Access
    - Performs Total Inspection at Layers 2-7
    - Prevents Application and O/S Damage/Downtime
- Infrastructure Protection – Protects from Worms, Viruses, Trojans, DDoS Attacks, SYN Floods, Traffic Anomalies
    - Protects Network Equipment Vulnerabilities
    - Protects Against Anomalous Traffic Behavior
    - Automatic Baselining
    - Rate Limit, Block, or Alert on Thresholds
    - Supports Custom IP filters, ACLs

Through this combination of detection capabilities, the Intrusion Prevention appliance is able to detect a variety of events. To establish a security rule set, each event must have a corresponding action associated with it. The resulting alerts and statistics generated by this existing security appliance were used to gauge the effectiveness and accuracy of our IDS system. The UnityOne appliance is able to perform the following actions when an event is detected.

- Permit and Notify – Allow packets to pass through the protected segment and generate a system alert.
- Permit, Notify, and Trace – Allow packets to pass, generate an alert, and save packets in tcpdump format for analysis.
- Block – drop packets generating the alerts and all remaining packets in the network flow.
- Block and Notify – drop packets in corresponding flow and generate a system alert.

- Block, Notify, and Trace – drop packets in corresponding flow, generate a system alert, and record packets in tcpdump format for further analysis.

The ability to create an environment allowing the launching of DOS attacks and scan and probe attacks while fully monitoring the desired segment in a production environment presents several challenges that we will examine in this section.

- Positioning the IDS system
- The threat of introducing a point of failure in the network segment
- The ability to launch attacks among normal traffic without affecting the performance of a production network
- The ability to launch DOS and Scan and Probe attacks in an environment where such traffic would be dropped by firewall and Intrusion Prevention devices
- The ability to originate DOS and scan and probe attacks from IP addresses outside the local segment to work with the normal operation of the IDS.

Whenever possible, the IDS should be allowed to receive and analyze the same traffic as that of existing systems. The primary reasons for this included assurance that unobstructed exposure to the production network's traffic, does not:

- Hinder the ability of the system to detect attacks when attack traffic is exposed to the network traffic of a production network.
- Cause the IDS to generate alerts resulting from exposure to benign traffic that the system has not been exposed to during its development.

Exposure to identical traffic was ensured by spanning the appropriate port on the network switch, leading to the segments only gateway. The resulting network traffic would include all packets entering or leaving the DMZ as well as all multicast traffic created within the DMZ's broadcast domain.

Because network switches normally extend the collision domain to a single interface, several solutions exist for allowing interface traffic to be monitored by an external sniffer, IDS or other analysis tool. In many high availability environments, network taps have been a preferred solution, allowing a dedicated in-line hardware solution that eliminated some of the problems traditionally associated with software based monitoring solutions like SPAN.

SPAN, or Switched Port Analyzer is a software feature in Cisco operating systems that allows traffic from a specified source, to be mirrored to a specified destination. Source and destination can be an interface, VLAN, or a remote location through the use of the RSPAN feature. Some relevant limitations of SPAN include:

- SPAN sessions will not mirror runt or giant packets
- Malformed packets may not be forwarded
- Increased use of switch buffers
- Degradation of switch performance

While the ability to view runts, giants and malformed packets that would otherwise be dropped by the receiving interface may be desirable in a troubleshooting scenario, it will not have any adverse affects in the environment shown above. Such packets would be dropped by the network firewall and/or Intrusion prevention device before the receiving spanned interface is reached. The existence of these types of packets can be verified by analyzing the counters in the spanned interface.

Current implementations of SPAN do not impact normal operation of the switch. Unless oversubscribed, newer implementations eliminate concerns of degradation of switch performance and buffer use. To determine the possibility of oversubscribing this switch, the following statistics are relevant [24]:

- Cisco Catalyst 3750 switch with standard IOS
- 32 Gbps switching fabric
- 20 of 24 active interfaces
- 12 active interfaces utilize a full 1gigabit/second full duplex link
- Spanned interface regularly reveals zero packet errors as seen below. Discarded packets on this interface could be an indication of overloaded buffers. This is an indication of an oversubscribed switch or interface.

show interfaces gigabitethernet 1/0/1 counters errors regularly reveals zero discarded packets

```
Port     Align-Err   FCS-Err  Xmit-Err   Rcv-Err UnderSize
Gi1/0/1      0          0         0         0         0

Port     Single-Col    Multi-Col Late-Col Excess-Col Carri-Sen    Runts   Giants
Gi1/0/1        0            0        0         0          0          0       0
```

Figure 2.2 Gigabit interface output.

Because the traditional limitations of SPAN do not apply to the testing environment, the use of a network tap was not deemed necessary. The Cisco Catalyst 3750 supports a total of two simultaneous span sessions, enabling the dedicated use of one session for testing without denying college staff of a second session for troubleshooting and testing. [24]

- monitor session 1 source interface gigabitethernet 1/0/1 both
  - Specifies the source port to be monitored. Both incoming and outgoing packets are selected for mirroring to the destination port
- monitor session 1 destination interface gigabitethernet 1/0/1

- Specifies the destination port of the monitoring session

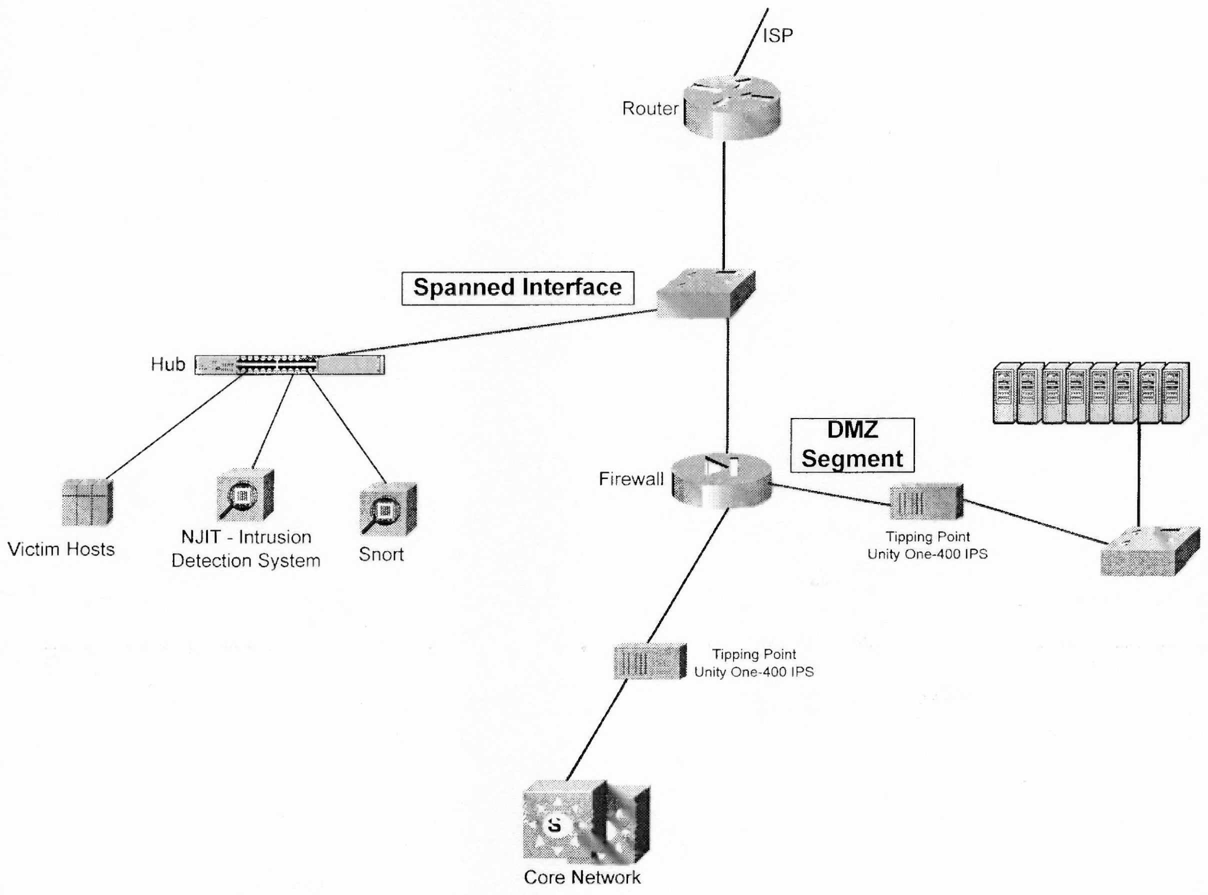

ISP
Router
Spanned Interface
Hub
DMZ Segment
Firewall
Victim Hosts
NJIT - Intrusion Detection System
Snort
Tipping Point Unity One-400 IPS
Tipping Point Unity One-400 IPS
Core Network

Figure 2.3 Modified test environment.

The location that would most aggressively allow us to gauge the performance of the IDS would be the external network segment. This segment would not be protected by a network firewall, allowing all scan attempts, DOS attacks, and related events to be monitored by the IDS system. The challenges of this design included:

- Difficulty in determining true alerts versus false alerts in a network segment with little to no restrictions

- Full exposure to internet viruses, worms, and scans. Numbers of alerts in such an environment is likely to be high.

- The network segment was not monitored by any existing intrusion detection or protection system

The challenges faced in this initial design proved excessively difficult to work with, forcing us to abandon the design when faced with unacceptably high false positive rates, unreliable training techniques, and difficulty in producing usable statistics. Instead, a more controlled and restricted environment was needed for initial testing. The existing demilitarized zone proved itself to be an ideal option for testing to continue.

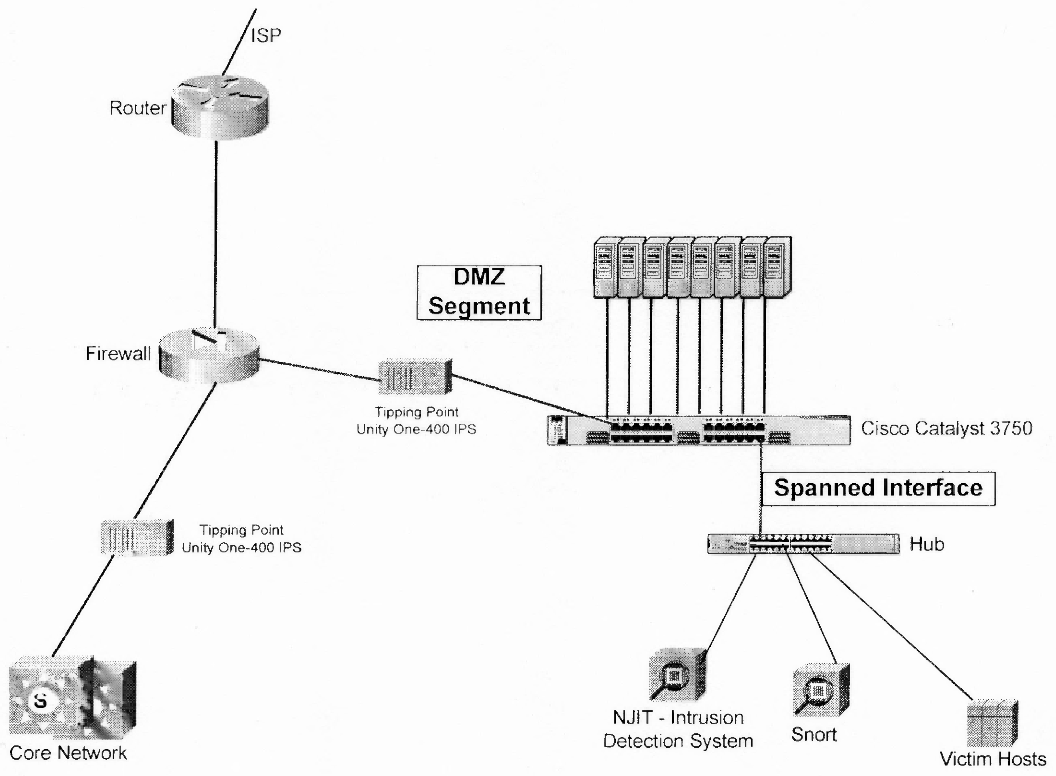


Figure 2.4 Final test environment.

The final testing environment creates a spanned segment, containing all traffic entering or leaving the DMZ segment. Traffic between individual servers in the DMZ segment will not be mirrored into the test environment. This is the traditional design of a monitored network segment in is sufficient for our testing purposes. An important fact in the creation of this environment is that the interface selected as the destination port for the monitoring session will not allow incoming packets to enter/re-enter the traditional

DMZ segment, allowing attacks to be launched in the spanned segment without interfering with normal college traffic or operations. Some limitations to the current testing environment include the following:

- The nature of the IDS system being tested limited monitored traffic flow to those originating from outside the local subnet, requiring the use of spoofed source addresses and/or the ability to pass traffic through firewalls and intrusion prevention devices.
- Strict limits on the volume of network traffic that can normally be passed across production network links
- Lack of remote communication to the tested IDS systems resulting from a signed agreement between the college and the thesis professor.

2.2 DOS Testing

In order to test the various intrusion detection algorithms available, tests performed were separated into two categories. The first series of tests consisted of controlled Denial of Service attacks of varying intensity

2.2.1 DOS Attacks

TCP SYN flood

A TCP SYN flood is indicated by an attempt to overwhelm a TCP service running on a victim machine by sending a large number of TCP SYN packets without completing the TCP handshake or resetting the connection.

A typical network server will contain a data structure of finite size in system memory describing pending connections. A SYN flood will attempt to overflow this data structure by creating large numbers of partially-open connections. Although these connections will eventually timeout, a TCP SYN flood will normally involve a high volume of network traffic from one or more attacking systems, overwhelming the victim machines, possibly resulting in exhausting system memory, system crash, or a system being rendered otherwise inoperative. [25] The potential of a SYN flood can be multiplied by the TCP/RST responses generated by the victim machine.

ICMP Flood

A simple ICMP flood was tested against victims by generating a high volume of ICMP echo requests. The attack will normally send large numbers of ICMP ping request packets with spoofed source IP addresses, leaving the victim with the need to reply with equally large number of response packets. An ICMP flood attempts to overload a system with so many echo requests that the system expends all resources responding, leaving it unable to process valid network traffic. [26]

UDP Flood

Another common category of attacks used in testing consists of a flood of UDP network traffic. When a connection is established between two UDP services, each producing network traffic, these two services can produce a very high number of packets that can lead to a denial of service on the machine(s) where the services are offered. For example, by connecting a host's chargen service to the echo service of another machine, all affected machines may be effectively taken out of service because of the excessively high number of packets produced. In addition, the intervening network may become congested and deny service to all hosts whose traffic traverses that network. [27]

A UDP Flood Attack can be created by simply transmitting UDP packets to a random port on a victim system. When the victim system receives each UDP packet, it will determine what application is waiting on the destination port. If there is no application waiting on the port, it will generate an ICMP packet of destination unreachable. In addition to flooding the victim with UDP traffic, this attack exploits the fact that for every UDP packet sent to a closed port, there will be an ICMP unreachable message sent back, multiplying the attacks potential. If enough UDP packets are delivered to ports on victim, the system can be rendered inoperative. [28]

SMURF

A smurf attack is created when an attacker generates ICMP echo request packets directed to IP broadcast addresses of an intermediary network. The broadcast destination address acts as an amplifier. The spoofed source address of these broadcast packets will become the victim of the attack. If the intermediary does not filter ICMP traffic directed to IP broadcast addresses, all machines on the network will receive this ICMP echo request. When these machines respond to the ICMP echo requests, they send replies to the victim's machine. The victim is subjected to network congestion that could potentially make the network unusable. Additionally, the intermediary networks can be classified as victims by suffering the same types of problem that the victim host receives. [29]

FRAGGLE

Fraggle, sometimes considered the cousin of the smurf attack, uses UDP echo packets in the same fashion as the ICMP echo packets are used in the smurf attack. It was a simple re-write of smurf that uses UDP echo and echo reply messages instead of ICMP messages. [30] Because UDP echoes typically are filtered, Fraggle typically has a less likelihood of having the same impact that Smurf does. [31]

BLOOP

Another simple denial of service flood attack is bloop. Bloop begins when an attacker sends a flood of ICMP unreachable messages to a victim host. The resulting effect will be a slowdown of network services and high CPU utilization on the victim required to deal with the overwhelming number of network requests. [32]

TEARDROP

An IP fragmentation attack takes advantage of the IP protocol's fragmentation feature to disrupt services of a remote host. While many variants exist, teardrop attacks take advantage of a weakness in some operating systems' ability to reassemble overlapping IP fragments. Teardrop variants include targa, SYNdrop, Boink, Nestea Bonk, TearDrop2 and NewTear. [33]

TARGA3

Targa3 attacks consist of a combination of IP based protocol packets with values known to be critical or bogus, causing some IP stack implementations to crash, fail, or show other undefined behavior. [34]

Targa3 works by sending uncommon IP packets to a victim host. Packets may consists of invalid fragment, protocol, packet size, or header values. When the victim TCP stack receives the invalid packet, the OS kernel had to allocate resources to handle the packet. When executed with a high number of network packets, the victim system would crash because of exhausted resource. [35]

To simulate and create the described denial of service attacks, a number of tools were used to generate the necessary traffic. The following tools were used either directly or to create an IP packet capture file that could be replayed at various levels of intensity to generate the desired attack traffic. All attack traffic would be presented to both the college IPS equipment as well as the intrusion detection systems being developed.

2.2.2 DOS Tools

TFN2k (Tribe FloodNet 2k edition)

Tribe FloodNet 2k, the current version of this attack tool offers a rich set of DOS tools that can enable an attacker to launch a variety attacks from the toolset or take control of a large distributed network of attack hosts.

TFN features in this version include:

- Remote one-way command execution for distributed execution control
- Mix attack aimed at weak routers
- Targa3 attack aimed at systems with IP stack vulnerabilities
- Compatibility to many UNIX systems and Windows NT
- Anonymous stealth client/server communication using:
    - spoofed source addresses
    - strong advanced encryption
    - one-way communication protocol
    - messaging via random IP protocol
    - decoy packets
- Attack Types available through this menu driven tool include UDP flood attack, SYN flood attack, ICMP echo reply (ping) attack, SMURF attack, and TARGA3 attack described above. Additionally, a MIX attack will send UDP, SYN and ICMP packets interchanged on a 1:1:1 relation. This attack can be hazardous to routers, NIDS and sniffers.[34]

Additionally, a number of unidentified attack tools gathered from hacking websites were obtained. Each tool was compiled and tested. The resulting network traffic was analyzed to verify the tool's functionality. Once verified, the tool was used in testing. In most cases, a number of tools were used to launch a tested attack type to ensure that results were not based on the effects of a specific tool rather than attack type.

2.2.3 DOS Tests Performed

Denial of service tests consisted of 16 types of tests. Each test type was run at four levels of intensity ranging from a low intensity test to a full flood utilizing all available bandwidth. Results of each instance were recorded against both the Conex IDS system and the college IPS appliance. Attacks consisted of the following:

- Two SYN flood attacks using hping and an anonymous SYN flood tool.
- Distributed SYN, UDP flood, ICMP flood, mix, smurf, and targa3 attacks using tfn2k.
- A distributed fraggle attack using an anonymous attack tool
- ICMP, UDP, and mix flood attacks using hping3.
- Smurf, bloop, teardrop, and fraggle attacks using anonymous tools.

2.2.4 DOS Test Results

Denial of service results varied at first. While Tipping Point was able to detect 100% of tests performed, the Conex IDS was unable to detect a small number of attacks. After some examination of the attacks performed, the Conex IDS system was retrained to include the new attack codes. When completed, the tests were repeated with much more positive results. Results are summarized in the Table 2.1 and Table 2.2

Table 2.1 DOS Results – Initial Test

Attack Type	Tool	Low	Med	High	Flood
SYN1 – IDS	SYN source	✗	✗	✗	✗
SYN1 – TP	SYN source	✓	✓	✓	✓
SYN2 – IDS	Hping3	✗	✗	✗	✗
SYN2 - TP	Hping3	✓	✓	✓	✓
Distributed SYN – IDS	TFN2K	✗	✗	✗	✗
Distributed SYN - TP	TFN2K	✓	✓	✓	✓
ICMP Flood – IDS	Hping3	✗	✗	✓	✓
ICMP Flood - TP	Hping3	✓	✓	✓	✓
Distributed ICMP Flood – IDS	TFN2K	✗	✗	✓	✓
Distributed ICMP Flood - TP	TFN2K	✓	✓	✓	✓
UDP Flood – IDS	Hping3	✓	✓	✓	✓
UDP Flood - TP	Hping3	✓	✓	✓	✓
Distributed UDP Flood – IDS	TFN2K	✗	✓	✓	✓
Distributed UDP Flood - TP	TFN2K	✓	✓	✓	✓
ICMP/UDP/SYN Flood – IDS	HPING3	✓	✓	✓	✓
ICMP/UDP/SYN Flood - TP	HPING3	✓	✓	✓	✓
Distributed MIX Flood – IDS	TFN2K	✓	✓	✓	✓
Distributed MIX Flood - TP	TFN2K	✓	✓	✓	✓
SMURF – IDS	Smurf source	✗	✗	✗	✓
SMURF - TP	Smurf source	✓	✓	✓	✓
Distributed SMURF – IDS	Smurf Source	✗	✗	✗	✓
Distributed SMURF - TP	Smurf Source	✓	✓	✓	✓
BLOOP – IDS	Bloop Source	✓	✓	✓	✓
BLOOP - TP	Bloop Source	✓	✓	✓	✓
TEARDROP – IDS	Teardrop Source	✓	✓	✓	✓
TEARDROP - TP	Teardrop Source	✓	✓	✓	✓
FRAGGLE – IDS	Fraggle Source	✓	✓	✓	✓
FRAGGLE - TP	Fraggle Source	✓	✓	✓	✓
Distributed Fraggle – IDS	Fraggle Source	✓	✓	✓	✓
Distributed Fraggle - TP	Fraggle Source	✓	✓	✓	✓
Distributed TARGA3 – IDS	TFN2K	✓	✓	✓	✓
Distributed TARGA3 - TP	TFN2K	✓	✓	✓	✓

Table 2.2 DOS Results – Final Test

Attack Type	Tool	Low	Med	High	Flood
SYN1 – IDS	SYN source	✓	✓	✓	✓
SYN1 – TP	SYN source	✓	✓	✓	✓
SYN2 – IDS	Hping3	✓	✓	✓	✓
SYN2 - TP	Hping3	✓	✓	✓	✓
Distributed SYN – IDS	TFN2K	×	✓	✓	✓
Distributed SYN - TP	TFN2K	✓	✓	✓	✓
ICMP Flood – IDS	Hping3	✓	✓	✓	✓
ICMP Flood - TP	Hping3	✓	✓	✓	✓
Distributed ICMP Flood – IDS	TFN2K	✓	✓	✓	✓
Distributed ICMP Flood - TP	TFN2K	✓	✓	✓	✓
UDP Flood – IDS	Hping3	✓	✓	✓	✓
UDP Flood - TP	Hping3	✓	✓	✓	✓
Distributed UDP Flood – IDS	TFN2K	×	✓	✓	✓
Distributed UDP Flood - TP	TFN2K	✓	✓	✓	✓
ICMP/UDP/SYN Flood – IDS	HPING3	✓	✓	✓	✓
ICMP/UDP/SYN Flood - TP	HPING3	✓	✓	✓	✓
Distributed MIX Flood – IDS	TFN2K	✓	✓	✓	✓
Distributed MIX Flood - TP	TFN2K	✓	✓	✓	✓
SMURF – IDS	Smurf source	✓	✓	✓	✓
SMURF - TP	Smurf source	✓	✓	✓	✓
Distributed SMURF – IDS	Smurf Source	✓	✓	✓	✓
Distributed SMURF - TP	Smurf Source	✓	✓	✓	✓
BLOOP – IDS	Bloop Source	✓	✓	✓	✓
BLOOP - TP	Bloop Source	✓	✓	✓	✓
TEARDROP – IDS	Teardrop Source	✓	✓	✓	✓
TEARDROP - TP	Teardrop Source	✓	✓	✓	✓
FRAGGLE – IDS	Fraggle Source	✓	✓	✓	✓
FRAGGLE - TP	Fraggle Source	✓	✓	✓	✓
Distributed Fraggle – IDS	Fraggle Source	✓	✓	✓	✓
Distributed Fraggle - TP	Fraggle Source	✓	✓	✓	✓
Distributed TARGA3 – IDS	TFN2K	✓	✓	✓	✓
Distributed TARGA3 - TP	TFN2K	✓	✓	✓	✓

2.3 Scan and Probe Testing

The next series of tests performed consisted of scan and probe attacks of varying intensity. Using various tools, a number of reconnaissance scanning techniques were used to compare the Conex system's detection capabilities to that of existing systems.

2.3.1 Scan and Probe Attacks

The following section describes the contents of eleven types of scan and probe attacks used to test the capabilities of the Conex system. The output of each tool was run and recorded as a TCP CAP file to ensure that the test was as simple as possible. A number of scan and probe tools were found to be excessively "noisy" when run at default settings. This would most likely lead to questionable results. In order to ensure that the detection of an IDS alert resulted only from the most basic scan and probe attack, the output from a number of tools were tested and the most basic was selected and described in the following section. Scan and probe attacks consisted of the following:

- ICMP Echo Request
- ICMP Info Request
- ICMP Netmask Request
- ICMP Timestamp Request
- TCP Connect and Half Connect
- SYN Stealth
- TCP FIN Connect
- TCP NULL Connect
- TCP XMAS Connect
- UDP Connect

2.3.1.1 ICMP Echo Request. A fundamental method of detecting the existence of a functional IP stack on a remote computer is to utilize ICMP Ping. Simple ICMP Ping request and reply packets are exchanged between two hosts as seen below. Many security conscious administrators may block ICMP packets at their border firewalls.

Otherwise, it is a simple method of detecting systems that provides the first steps of reconnaissance to a scanner.

ICMP can prove very effective when not blocked by corporate firewall policy. The presence of ICMP packets traversing a network is most often ignored by intrusion detection systems unless combined with other more revealing types of traffic.

While most implementations of PING require a timeout period, dozens of network scanners exist that allow for ping sweeps across an IP subnet and very strong control of ICMP packet speeds, sometimes enough to provide a primitive form of DOS.

```
[root@localhost jleon]# /usr/sbin/hping2 130.156.1.15 -1
HPING 130.156.1.15 (eth0 130.156.1.15): icmp mode set, 28 headers + 0 data byteslen=46 ip=130.156.1.15 ttl=52 id=41873 icmp_seq=0 rtt=143.7
ms
len=46 ip=130.156.1.15 ttl=52 id=41874 icmp_seq=1 rtt=26.0 ms
len=46 ip=130.156.1.15 ttl=52 id=41875 icmp_seq=2 rtt=16.6 ms
len=46 ip=130.156.1.15 ttl=52 id=41876 icmp_seq=3 rtt=13.5 ms
len=46 ip=130.156.1.15 ttl=52 id=41877 icmp_seq=4 rtt=17.8 ms
len=46 ip=130.156.1.15 ttl=52 id=41878 icmp_seq=5 rtt=37.4 ms

--- 130.156.1.15 hping statistic ---
6 packets tramitted, 6 packets received, 0% packet loss
round-trip min/avg/max = 13.5/42.5/143.7 ms
```

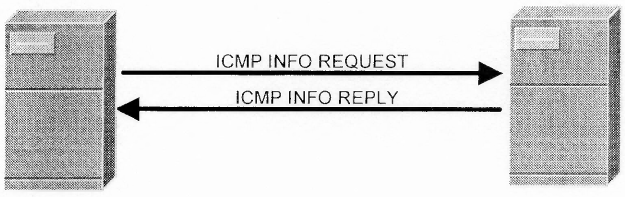
PING REQUEST
PING REPLY
PING REQUEST
PING REPLY

```
Frame 2 (42 bytes on wire, 42 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.120 (192.168.1.120), Dst Addr: 130.156.1.15 (130.156.1.15)
Internet Control Message Protocol
   Type: 8 (Echo (ping) request)
   Sequence number: 0x0000

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 1c b0 db 00 00 40 01 84 3a c0 a8 01 78 82 9c   ......@..:...x..
0020  01 0f 08 00 4d b3 aa 4c 00 00                     ....M..L..

Frame 3 (60 bytes on wire, 60 bytes captured)
Ethernet II, Src: 00:06:25:80:db:93, Dst: 00:b0:d0:05:43:ec
Internet Protocol, Src Addr: 130.156.1.15 (130.156.1.15), Dst Addr: 192.168.1.120 (192.168.1.120)
Internet Control Message Protocol
   Type: 0 (Echo (ping) reply)
   Sequence number: 0x0000

0000  00 b0 d0 05 43 ec 00 06 25 80 db 93 08 00 45 00   ....C...%.....E.
0010  00 1c a3 91 00 00 34 01 9d 84 82 9c 01 0f c0 a8   ......4.........
0020  01 78 00 00 55 b3 aa 4c 00 00 00 00 00 00 00 00   .x..U..L........
0030  00 00 00 00 00 00 00 00 00 00 00 00               ............

Frame 4 (42 bytes on wire, 42 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.120 (192.168.1.120), Dst Addr: 130.156.1.15 (130.156.1.15)
Internet Control Message Protocol
   Type: 8 (Echo (ping) request)
   Sequence number: 0x0100

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 1c e1 1a 00 00 40 01 53 fb c0 a8 01 78 82 9c   ......@.S....x..
0020  01 0f 08 00 4c b3 aa 4c 01 00                     ....L..L..

Frame 5 (60 bytes on wire, 60 bytes captured)
Ethernet II, Src: 00:06:25:80:db:93, Dst: 00:b0:d0:05:43:ec
Internet Protocol, Src Addr: 130.156.1.15 (130.156.1.15), Dst Addr: 192.168.1.120 (192.168.1.120)
Internet Control Message Protocol
   Type: 0 (Echo (ping) reply)
   Sequence number: 0x0100

0000  00 b0 d0 05 43 ec 00 06 25 80 db 93 08 00 45 00   ....C...%.....E.
0010  00 1c a3 92 00 00 34 01 9d 83 82 9c 01 0f c0 a8   ......4.........
0020  01 78 00 00 54 b3 aa 4c 01 00 00 00 00 00 00 00   .x..T..L........
0030  00 00 00 00 00 00 00 00 00 00 00 00               ............
```

Figure 2.5 ICMP echo request.

2.3.1.2 ICMP Info Request. An alternative to ICMP PING packets that may be dropped is the ICMP Information request. Generating an automatic response, it can be used to test the existence of a system.

```
[root@localhost jleon]# sing -v -info 130.156.1.15 -c 1
SINGing to 130.156.1.15 (130.156.1.15): 8 data bytes
8 bytes from 130.156.1.15: seq=0 ttl=52 TOS=0 Info Reply


--- 130.156.1.15 sing statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
```

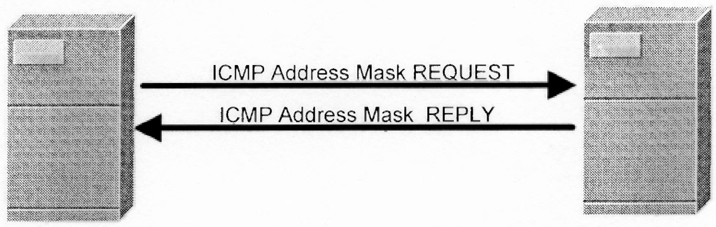


```
Frame 1 (42 bytes on wire, 42 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.120 (192.168.1.120), Dst Addr: 130.156.1.15 (130.156.1.15)
Internet Control Message Protocol
  Type: 15 (Information request)
  Code: 0
  Checksum: 0x73ef (correct)
  Identifier: 0x7d10
  Sequence number: 0x0000

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 1c 33 72 00 00 ff 01 42 a3 c0 a8 01 78 82 9c   ..3r....B....x..
0020  01 0f 0f 00 73 ef 7d 10 00 00                     ....s.}...

Frame 2 (60 bytes on wire, 60 bytes captured)
Ethernet II, Src: 00:06:25:80:db:93, Dst: 00:b0:d0:05:43:ec
Internet Protocol, Src Addr: 130.156.1.15 (130.156.1.15), Dst Addr: 192.168.1.120 (192.168.1.120)
Internet Control Message Protocol
  Type: 16 (Information reply)
  Code: 0
  Checksum: 0x72ef (correct)
  Identifier: 0x7d10
  Sequence number: 0x0000

0000  00 b0 d0 05 43 ec 00 06 25 80 db 93 08 00 45 00   ....C...%.....E.
0010  00 1c 91 52 00 00 34 01 af c3 82 9c 01 0f c0 a8   ...R..4.........
0020  01 78 10 00 72 ef 7d 10 00 00 00 00 00 00 00 00   .x..r.}.........
0030  00 00 00 00 00 00 00 00 00 00 00 00               ............
```

Figure 2.6 ICMP info request.

2.3.1.3 ICMP Netmask Request. ICMP Network Mask requests are another alternative that can be used to both reveal the existence of a functional node and its subnet mask when other ICMP packets may be blocked by a firewall.

```
[root@localhost jleon]# sing -mask -c 1 -v 198.242.56.30
SINGing to 198.242.56.30 (198.242.56.30): 12 data bytes
12 bytes from 198.242.56.30: seq=0 DF! ttl=246 TOS=0 mask=255.255.255.0
```



```
--- 198.242.56.30 sing statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
```

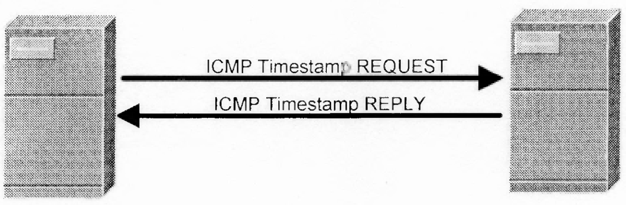
```
Frame 1 (46 bytes on wire, 46 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.120 (192.168.1.120), Dst Addr: 198.242.56.30 (198.242.56.30)
Internet Control Message Protocol
   Type: 17 (Address mask request)

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 20 33 72 00 00 ff 01 c7 39 c0 a8 01 78 c6 f2   . 3r.....9...x..
0020  38 1e 11 00 c9 f3 25 0c 00 00 00 00 00 00         8.....%.......

Frame 2 (60 bytes on wire, 60 bytes captured)
Ethernet II, Src: 00:06:25:80:db:93, Dst: 00:b0:d0:05:43:ec
Internet Protocol, Src Addr: 198.242.56.30 (198.242.56.30), Dst Addr: 192.168.1.120 (192.168.1.120)
Internet Control Message Protocol
   Type: 18 (Address mask reply)
   Address mask: 255.255.255.0 (0xffffff00)

0000  00 b0 d0 05 43 ec 00 06 25 80 db 93 08 00 45 00   ....C...%.....E.
0010  00 20 51 4b 40 00 f6 01 72 60 c6 f2 38 1e c0 a8   . QK@...r`..8...
0020  01 78 12 00 c9 f2 25 0c 00 00 ff ff ff 00 00 00   .x....%.........
0030  00 00 00 00 00 00 00 00 00 00 00 00               ............
```

Figure 2.6 ICMP netmask request.

2.3.1.4 ICMP Timestamp Request. Another ICMP alternative is the timestamp function. Generating an automatic response, it can be used to test the existence of a system when other ICMP traffic types may be blocked.

Below is an example of an ICMP request and response using the SING (Send ICMP Network Garbage) utility.

```
[root@localhost jleon]# sing -v -tstamp 130.156.1.15 -c 1
SINGing to 130.156.1.15 (130.156.1.15): 20 data bytes
20 bytes from 130.156.1.15: seq=0 ttl=52 TOS=0 diff=34341565


--- 130.156.1.15 sing statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
```



```
Frame 1 (54 bytes on wire, 54 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.120 (192.168.1.120), Dst Addr: 130.156.1.15 (130.156.1.15)
Internet Control Message Protocol
   Type: 13 (Timestamp request)
   Code: 0
   Checksum: 0x82e2 (correct)
   Identifier: 0x3110
   Sequence number: 0x0000
   Originate timestamp: 23281066
   Receive timestamp: 0
   Transmit timestamp: 0

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 28 33 72 00 00 ff 01 42 97 c0 a8 01 78 82 9c   .(3r....B....x..
0020  01 0f 0d 00 82 e2 31 10 00 00 01 63 3d aa 00 00   ......1....c=...
0030  00 00 00 00 00 00                                 ......

Frame 2 (60 bytes on wire, 60 bytes captured)
Ethernet II, Src: 00:06:25:80:db:93, Dst: 00:b0:d0:05:43:ec
Internet Protocol, Src Addr: 130.156.1.15 (130.156.1.15), Dst Addr: 192.168.1.120 (192.168.1.120)
Internet Control Message Protocol
   Type: 14 (Timestamp reply)
   Code: 0
   Checksum: 0xfa35 (correct)
   Identifier: 0x3110
   Sequence number: 0x0000
   Originate timestamp: 23281066
   Receive timestamp: 57622631
   Transmit timestamp: 57622631

0000  00 b0 d0 05 43 ec 00 06 25 80 db 93 08 00 45 00   ....C...%.....E.
0010  00 28 87 26 00 00 34 01 b9 e3 82 9c 01 0f c0 a8   .(.&..4.........
0020  01 78 0e 00 fa 35 31 10 00 00 01 63 3d aa 03 6f   .x...51....c=..o
0030  40 67 03 6f 40 67 00 00 00 00 00 00               @g.o@g......
```

Figure 2.8 ICMP timestamp request.

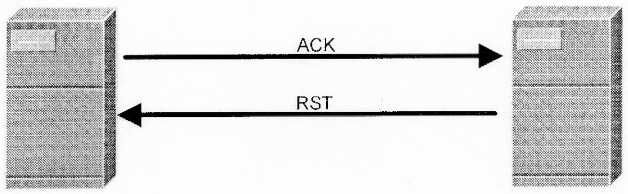
2.3.1.5 TCP Half Connect. An alternative method of detecting the presence of an IP stack on a remote host without the use of ICMP ping is to attempt a TCP connection. Easily detectable by local system logs, firewalls, and IDS systems, it is common to attempt connections to common TCP ports without following the TCP handshake. Instead, an ACK packet is sent to a host. Internet standards require a reset be sent in response to a TCP ACK without the proper completion of a TCP handshake. While

Microsoft operating systems do not behave this way, TCP ACK packets are an effective alternative to PING.

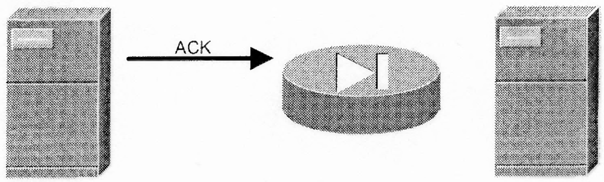
This method does not reveal the status of the port (open, closed) because the end system will send a reset regardless of the state. Also, stateful firewalls will not respond to these requests at all and will instead generate errors easily detectable by firewall administrators as shown below.

The example below utilizes the HPING 2 utility to attempt a single connection to port 21 on a remote host.

```
[root@localhost jleon]# /usr/sbin/hping2 -A -p 21 130.156.1.15 -V -c 1
using eth0, addr: 192.168.1.120, MTU: 1500
HPING 130.156.1.15 (eth0 130.156.1.15): A set, 40 headers + 0 data bytes
len=46 ip=130.156.1.15 ttl=48 id=44967 tos=20 iplen=40
sport=21 flags=R seq=0 win=32768 rtt=135.4 ms
seq=1403374968 ack=0 sum=8bbe urp=0

--- 130.156.1.15 hping statistic ---
1 packets tramitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 135.4/135.4/135.4 ms
```



```
Frame 1 (54 bytes on wire, 54 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.120 (192.168.1.120), Dst Addr: 130.156.1.15 (130.156.1.15)
Transmission Control Protocol, Src Port: 2646 (2646), Dst Port: ftp (21), Seq: 0, Ack: 0, Len: 0
   Flags: 0x0010 (ACK)

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 28 02 e3 00 00 40 06 32 22 c0 a8 01 78 82 9c   .(....@.2"...x..
0020  01 0f 0a 56 00 15 16 bd 12 d2 53 a5 cd 78 50 10   ...V......S..xP.
0030  02 00 12 f1 00 00                                 ......

Frame 2 (60 bytes on wire, 60 bytes captured)
Ethernet II, Src: 00:06:25:80:db:93, Dst: 00:b0:d0:05:43:ec
Internet Protocol, Src Addr: 130.156.1.15 (130.156.1.15), Dst Addr: 192.168.1.120 (192.168.1.120)
Transmission Control Protocol, Src Port: ftp (21), Dst Port: 2646 (2646), Seq: 0, Ack: 3913477422, Len: 0
   Flags: 0x0004 (RST)

0000  00 b0 d0 05 43 ec 00 06 25 80 db 93 08 00 45 20   ....C...%.....E
0010  00 28 af a7 00 00 30 06 95 3d 82 9c 01 0f c0 a8   .(....0..=......
0020  01 78 00 15 0a 56 53 a5 cd 78 00 00 00 00 50 04   .x...VS..x....P.
0030  80 00 be 8b 00 00 00 00 00 00 00 00               ............


[root@localhost jleon]# /usr/sbin/hping2 -A -p 80,21 130.156.1.4 -V -c 1
using eth0, addr: 192.168.1.120, MTU: 1500
HPING 130.156.1.4 (eth0 130.156.1.4): A set, 40 headers + 0 data bytes

--- 130.156.1.4 hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```



```
Frame 1 (54 bytes on wire, 54 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.120 (192.168.1.120), Dst Addr: 130.156.1.4 (130.156.1.4)
Transmission Control Protocol, Src Port: 2882 (2882), Dst Port: http (80), Seq: 0, Ack: 0, Len: 0
   Flags: 0x0010 (ACK)

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 28 fc d5 00 00 40 06 38 3a c0 a8 01 78 82 9c   .(....@.8:...x..
0020  01 04 0b 42 00 50 7d 77 a6 c3 51 49 84 39 50 10   ...B.P}w..QI.9P.
0030  02 00 62 c4 00 00                                 ..b..

Firewall SYSLOG Result of above:
Oct 17 2004 00:34:49: %PIX-6-106015: Deny TCP (no connection) from 68.162.6.253/50225 to 130.156.1.4/80 flags RST  on interface outside
```

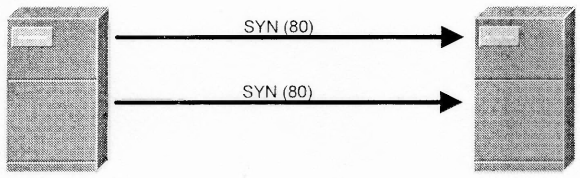
Figure 2.9 TCP half connect.

2.3.1.6 TCP Connect. A full TCP connect scan reliably reveals the state of each scanned port on an IP enabled host. It is, however, easy to detect on firewalls, IDS systems, and host logs. A more stealthy solution is a TCP SYN stealth scan.

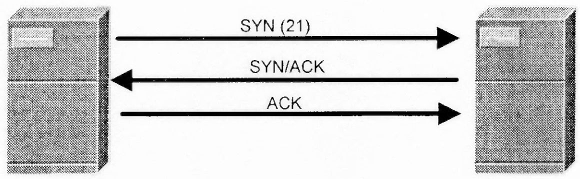
Below is an example of nmap performing this type of scan on three separate ports of a single host, identifying the three possible states and the behavior exhibited by a host in each of these states.

```
nmap -sT -p 80,21,11 130.156.1.29 -v -P0 -n -v

Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at 2004-09-27 21:15 EDT Initiating Connect() Scan against 130.156.1.29 [3 ports] at 21:15
Discovered open port 21/tcp on 130.156.1.29 The Connect() Scan took 12.18s to scan 3 total ports.
Host 130.156.1.29 appears to be up ... good.
Interesting ports on 130.156.1.29:
PORT   STATE   SERVICE
11/tcp closed  systat
21/tcp open    ftp
80/tcp filtered http

Nmap run completed -- 1 IP address (1 host up) scanned in 12.193 seconds
```

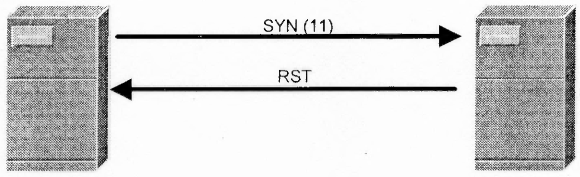
```
Frame 1 (74 bytes on wire, 74 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.123 (192.168.1.123), Dst Addr: 130.156.1.29 (130.156.1.29)
Transmission Control Protocol, Src Port: 1054 (1054), Dst Port: http (80), Seq: 0, Ack: 0, Len: 0
   Flags: 0x0002 (SYN)

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 3c 15 64 40 00 40 06 df 7b c0 a8 01 7b 82 9c   .<.d@.@..{...{..
0020  01 1d 04 1e 00 50 30 f9 0c b6 00 00 00 00 a0 02   .....P0.........
0030  16 d0 8f 67 00 00 02 04 05 b4 04 02 08 0a 00 04   ...g............
0040  19 d2 00 00 00 00 01 03 03 00                     ..........
```



```
Frame 2 (74 bytes on wire, 74 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.123 (192.168.1.123), Dst Addr: 130.156.1.29 (130.156.1.29)
Transmission Control Protocol, Src Port: 1055 (1055), Dst Port: ftp (21), Seq: 0, Ack: 0, Len: 0
   Flags: 0x0002 (SYN)

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 3c dd ab 40 00 40 06 17 34 c0 a8 01 7b 82 9c   .<..@.@..4...{..
0020  01 1d 04 1f 00 15 31 27 c4 79 00 00 00 00 a0 02   ......1'.y......
0030  16 d0 d7 af 00 00 02 04 05 b4 04 02 08 0a 00 04   ................
0040  19 d2 00 00 00 00 01 03 03 00                     ..........
```



```
Frame 3 (74 bytes on wire, 74 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.123 (192.168.1.123), Dst Addr: 130.156.1.29 (130.156.1.29)
Transmission Control Protocol, Src Port: 1056 (1056), Dst Port: systat (11), Seq: 0, Ack: 0, Len: 0
   Flags: 0x0002 (SYN)

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 3c f0 a9 40 00 40 06 04 36 c0 a8 01 7b 82 9c   .<..@.@..6...{..
0020  01 1d 04 20 00 0b 31 3e e9 7b 00 00 00 00 a0 02   ... ..1>.{......
0030  16 d0 b2 9f 00 00 02 04 05 b4 04 02 08 0a 00 04   ................
0040  19 d2 00 00 00 00 01 03 03 00                     ..........
```



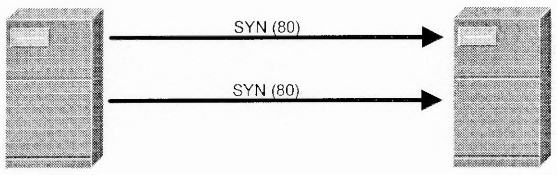
```
Frame 4 (60 bytes on wire, 60 bytes captured)
Ethernet II, Src: 00:06:25:80:db:93, Dst: 00:b0:d0:05:43:ec
Internet Protocol, Src Addr: 130.156.1.29 (130.156.1.29), Dst Addr: 192.168.1.123 (192.168.1.123)
Transmission Control Protocol, Src Port: ftp (21), Dst Port: 1055 (1055), Seq: 0, Ack: 1, Len: 0
   Flags: 0x0012 (SYN, ACK)

0000  00 b0 d0 05 43 ec 00 06 25 80 db 93 08 00 45 80   ....C...%.....E.
0010  00 2c 8c 78 00 00 2f 06 b8 f7 82 9c 01 1d c0 a8   .,.x../.........
0020  01 7b 00 15 04 1f 09 88 8e 65 31 27 c4 7a 60 12   .{.......e1'.z`.
0030  02 18 c1 fa 00 00 02 04 02 18 00 00               ............

Frame 5 (54 bytes on wire, 54 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.123 (192.168.1.123), Dst Addr: 130.156.1.29 (130.156.1.29)
Transmission Control Protocol, Src Port: 1055 (1055), Dst Port: ftp (21), Seq: 1, Ack: 1, Len: 0
   Flags: 0x0010 (ACK)
....
```

Figure 2.10 TCP connect.

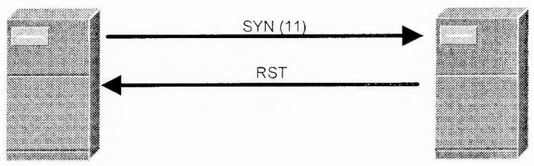
2.3.1.7 SYN Stealth. A TCP SYN stealth scan offers reliable results on the state of scanned ports with a higher degree of stealth. Because full connections are not made, a scanner is able to gain the same level of reliability in results as a full TCP Connect scan without the logging and detectability of these scans because the three way handshake is not completed.

Below is the same nmap scan performed in the full TCP scan using the stealthier TCP SYN alternate.

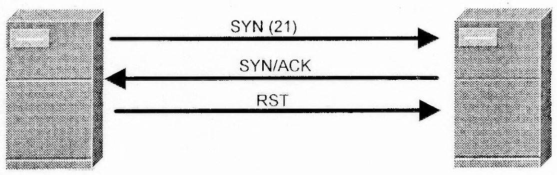
```
nmap -sS -p 80,21,11 130.156.1.29 -v -P0 -n -v

Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at 2004-09-27 21:17 EDT Initiating SYN Stealth Scan against 130.156.1.29 [3 ports] at 21:17
Discovered open port 21/tcp on 130.156.1.29 The SYN Stealth Scan took 1.31s to scan 3 total ports.
Host 130.156.1.29 appears to be up ... good.
Interesting ports on 130.156.1.29:
PORT   STATE   SERVICE
11/tcp closed  systat
21/tcp open    ftp
80/tcp filtered http

Nmap run completed -- 1 IP address (1 host up) scanned in 1.449 seconds
```



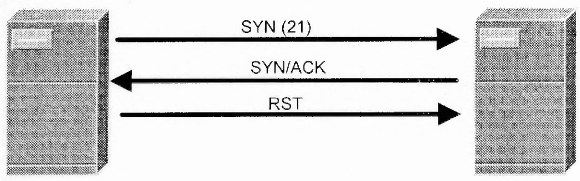
```
Frame 1 (54 bytes on wire, 54 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.123 (192.168.1.123), Dst Addr: 130.156.1.29 (130.156.1.29)
Transmission Control Protocol, Src Port: 61659 (61659), Dst Port: ftp (21), Seq: 0, Ack: 0, Len: 0
   Flags: 0x0002 (SYN)

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 28 23 f9 00 00 3b 06 15 fb c0 a8 01 7b 82 9c   .(#...;......{..
0020  01 1d f0 db 00 15 34 ca 95 15 00 00 00 00 50 02   ......4.......P.
0030  10 00 9f 35 00 00                                 ...5..
```



```
Frame 2 (54 bytes on wire, 54 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.123 (192.168.1.123), Dst Addr: 130.156.1.29 (130.156.1.29)
Transmission Control Protocol, Src Port: 61659 (61659), Dst Port: http (80), Seq: 0, Ack: 0, Len: 0
   Flags: 0x0002 (SYN)

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 28 31 c3 00 00 32 06 11 31 c0 a8 01 7b 82 9c   .(1...2..1...{..
0020  01 1d f0 db 00 50 34 ca 95 15 00 00 00 00 50 02   .....P4.......P.
0030  0c 00 a2 fa 00 00                                 ......
```



```
Frame 3 (54 bytes on wire, 54 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.123 (192.168.1.123), Dst Addr: 130.156.1.29 (130.156.1.29)
Transmission Control Protocol, Src Port: 61659 (61659), Dst Port: systat (11), Seq: 0, Ack: 0, Len: 0
   Flags: 0x0002 (SYN)

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 28 7f b9 00 00 2c 06 c9 3a c0 a8 01 7b 82 9c   .(....,..:...{..
0020  01 1d f0 db 00 0b 34 ca 95 15 00 00 00 00 50 02   ......4.......P.
0030  04 00 ab 3f 00 00                                 ...?..

Frame 4 (60 bytes on wire, 60 bytes captured)
Ethernet II, Src: 00:06:25:80:db:93, Dst: 00:b0:d0:05:43:ec
Internet Protocol, Src Addr: 130.156.1.29 (130.156.1.29), Dst Addr: 192.168.1.123 (192.168.1.123)
Transmission Control Protocol, Src Port: ftp (21), Dst Port: 61659 (61659), Seq: 0, Ack: 1, Len: 0
   Flags: 0x0012 (SYN, ACK)

0000  00 b0 d0 05 43 ec 00 06 25 80 db 93 08 00 45 80   ....C...%.....E.
0010  00 2c 8c 7d 00 00 2f 06 b8 f2 82 9c 01 1d c0 a8   .,.}../.........
0020  01 7b 00 15 f0 db 00 4e ba 3d 34 ca 95 16 60 12   .{.....N.=4...`.
0030  02 18 de 60 00 00 02 04 02 18 00 00               ...`........
```



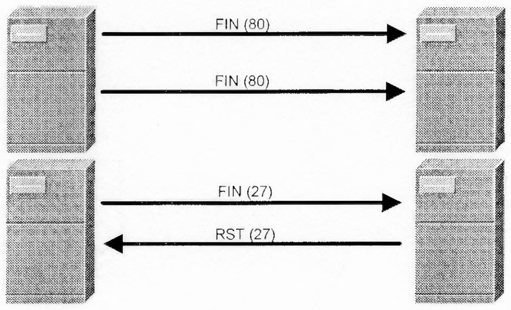
```
Frame 5 (54 bytes on wire, 54 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.123 (192.168.1.123), Dst Addr: 130.156.1.29 (130.156.1.29)
Transmission Control Protocol, Src Port: 61659 (61659), Dst Port: ftp (21), Seq: 1, Ack: 4289807811, Len: 0
   Flags: 0x0004 (RST)

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 80   ..%.......C...E.
0010  00 28 00 00 40 00 40 06 f4 73 c0 a8 01 7b 82 9c   .(..@.@..s...{..
0020  01 1d f0 db 00 15 34 ca 95 16 00 00 00 00 50 04   ......4.......P.
0030  00 00 af 32 00 00                                 ...2..
```

Figure 2.11 SYN stealth.

2.3.1.8 TCP FIN Connect. FIN Scans are TCP port scans identified by setting the FIN flag on the probe packets. RFC 793 states that systems should respond with a RST flag for a closed port.

Below is an example of a FIN scan on a Unix host with nmap. The lack of response indicates an open port as seen below.

```
[root@localhost jleon]# nmap -sF -v -n -P0 -p80,27 130.156.1.15

Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at 2004-09-30 11:13 EDT
Initiating FIN Scan against 130.156.1.15 [2 ports] at 11:13
Discovered open port 80/tcp on 130.156.1.15
The FIN Scan took 1.23s to scan 2 total ports.
Host 130.156.1.15 appears to be up ... good.
Interesting ports on 130.156.1.15:
PORT   STATE  SERVICE
27/tcp closed nsw-fe
80/tcp open   http


Nmap run completed -- 1 IP address (1 host up) scanned in 1.329 seconds
```



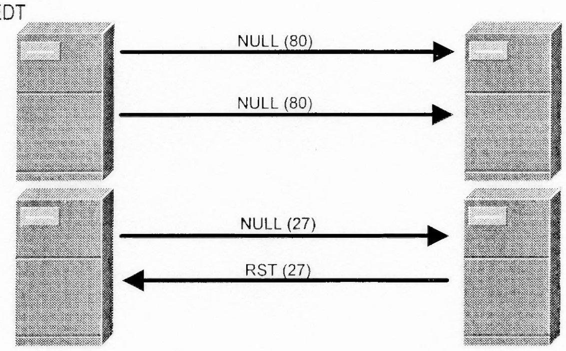
```
Frame 1 (54 bytes on wire, 54 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.120 (192.168.1.120), Dst Addr: 130.156.1.15 (130.156.1.15)
Transmission Control Protocol, Src Port: 61968 (61968), Dst Port: http (80), Seq: 0, Ack: 0, Len: 0
  Flags: 0x0001 (FIN)

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 28 6a 80 00 00 30 06 da 84 c0 a8 01 78 82 9c   .(j...0......x..
0020  01 0f f2 10 00 50 ba b9 ea 9b 00 00 00 00 50 01   .....P........P.
0030  04 00 ce 61 00 00                                 ...a..

Frame 2 (54 bytes on wire, 54 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.120 (192.168.1.120), Dst Addr: 130.156.1.15 (130.156.1.15)
Transmission Control Protocol, Src Port: 61968 (61968), Dst Port: 27 (27), Seq: 0, Ack: 0, Len: 0
  Flags: 0x0001 (FIN)

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 28 b1 33 00 00 36 06 8d d1 c0 a8 01 78 82 9c   .(.3..6......x..
0020  01 0f f2 10 00 1b ba b9 ea 9b 00 00 00 00 50 01   ..............P.
0030  0c 00 c6 96 00 00                                 ......

Frame 3 (60 bytes on wire, 60 bytes captured)
Ethernet II, Src: 00:06:25:80:db:93, Dst: 00:b0:d0:05:43:ec
Internet Protocol, Src Addr: 130.156.1.15 (130.156.1.15), Dst Addr: 192.168.1.120 (192.168.1.120)
Transmission Control Protocol, Src Port: 27 (27), Dst Port: 61968 (61968), Seq: 0, Ack: 0, Len: 0
  Flags: 0x0014 (RST, ACK)

0000  00 b0 d0 05 43 ec 00 06 25 80 db 93 08 00 45 00   ....C...%.....E.
0010  00 28 36 11 00 00 30 06 0e f4 82 9c 01 0f c0 a8   .(6...0.........
0020  01 78 00 1b f2 10 00 00 00 00 ba b9 ea 9b 50 14   .x............P.
0030  00 00 d2 83 00 00 00 00 00 00 00 00               ............

Frame 4 (54 bytes on wire, 54 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.120 (192.168.1.120), Dst Addr: 130.156.1.15 (130.156.1.15)
Transmission Control Protocol, Src Port: 61969 (61969), Dst Port: http (80), Seq: 0, Ack: 0, Len: 0
  Flags: 0x0001 (FIN)

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 28 7a 2a 00 00 3b 06 bf da c0 a8 01 78 82 9c   .(z*..;......x..
0020  01 0f f2 11 00 50 ba b8 ea 9a 00 00 00 00 50 01   .....P........P.
0030  10 00 c2 62 00 00                                 ...b..
```

Figure 2.12 TCP fin connect.

2.3.1.9 TCP NULL Connect. NULL Scans are TCP port scans identified by the absence of any flags on the TCP probe packets. RFC 793 states that systems should respond with a RST flag for a closed port.

Below is an example of a NULL scan on a Unix host with nmap. The lack of response indicates an open port as seen below.

```
[root@localhost jleon]# nmap -sN -v -n -P0 -p80,27 130.156.1.15
Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at 2004-09-30 11:20 EDT
Initiating NULL Scan against 130.156.1.15 [2 ports] at 11:20
Discovered open port 80/tcp on 130.156.1.15
The NULL Scan took 1.23s to scan 2 total ports.
Host 130.156.1.15 appears to be up ... good.
Interesting ports on 130.156.1.15:
PORT   STATE  SERVICE
27/tcp closed nsw-fe
80/tcp open   http


Nmap run completed -- 1 IP address (1 host up) scanned in 1.421 seconds
```

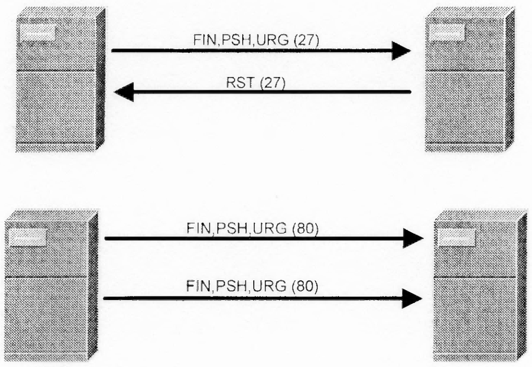


```
Frame 1 (54 bytes on wire, 54 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.120 (192.168.1.120), Dst Addr: 130.156.1.15 (130.156.1.15)
Transmission Control Protocol, Src Port: 35834 (35834), Dst Port: http (80), Seq: 0, Ack: 0, Len: 0
   Flags: 0x0000 ()

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 28 c1 97 00 00 29 06 8a 6d c0 a8 01 78 82 9c   .(....)..m...x..
0020  01 0f 8b fa 00 50 3b 6a 1d 3b 00 00 00 00 50 00   .....P;j.;....P.
0030  08 00 7d 29 00 00                                 ..})..

Frame 2 (54 bytes on wire, 54 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.120 (192.168.1.120), Dst Addr: 130.156.1.15 (130.156.1.15)
Transmission Control Protocol, Src Port: 35834 (35834), Dst Port: 27 (27), Seq: 0, Ack: 0, Len: 0
   Flags: 0x0000 ()

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 28 6f 0f 00 00 25 06 e0 f5 c0 a8 01 78 82 9c   .(o...%......x..
0020  01 0f 8b fa 00 1b 3b 6a 1d 3b 00 00 00 00 50 00   ......;j.;....P.
0030  08 00 7d 5e 00 00                                 ..}^..

Frame 3 (60 bytes on wire, 60 bytes captured)
Ethernet II, Src: 00:06:25:80:db:93, Dst: 00:b0:d0:05:43:ec
Internet Protocol, Src Addr: 130.156.1.15 (130.156.1.15), Dst Addr: 192.168.1.120 (192.168.1.120)
Transmission Control Protocol, Src Port: 27 (27), Dst Port: 35834 (35834), Seq: 0, Ack: 0, Len: 0
   Flags: 0x0014 (RST, ACK)

0000  00 b0 d0 05 43 ec 00 06 25 80 db 93 08 00 45 00   ....C...%.....E.
0010  00 28 3e 31 00 00 30 06 06 d4 82 9c 01 0f c0 a8   .(>1..0.........
0020  01 78 00 1b 8b fa 00 00 00 00 3b 6a 1d 3b 50 14   .x........;j.;P.
0030  00 00 85 4a 00 00 00 00 00 00 00 00               ...J........

Frame 4 (54 bytes on wire, 54 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.120 (192.168.1.120), Dst Addr: 130.156.1.15 (130.156.1.15)
Transmission Control Protocol, Src Port: 35835 (35835), Dst Port: http (80), Seq: 0, Ack: 0, Len: 0
   Flags: 0x0000 ()

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 28 f7 11 00 00 2b 06 52 f3 c0 a8 01 78 82 9c   .(....+.R....x..
0020  01 0f 8b fb 00 50 3b 6b 1d 3a 00 00 00 00 50 00   .....P;k.:....P.
0030  10 00 75 28 00 00
```

Figure 2.12 TCP null connect.

2.3.1.10 TCP XMAS Connect. Below is an example of an XMAS scan on a Unix host with nmap. XMAS Scans are TCP port scans identified by setting the FIN, URG, and PUSH flags on the probe packets. RFC 793 states that systems should respond with a RST flag for a closed port. The lack of response indicates an open port as seen below.

```
[root@localhost jleon]# nmap -sX -v -n -P0 -p80,27 130.156.1.15

Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at 2004-09-30 11:17 EDT
Initiating XMAS Scan against 130.156.1.15 [2 ports] at 11:17
Discovered open port 80/tcp on 130.156.1.15
The XMAS Scan took 1.25s to scan 2 total ports.
Host 130.156.1.15 appears to be up ... good.
Interesting ports on 130.156.1.15:
PORT   STATE  SERVICE
27/tcp closed nsw-fe
80/tcp open   http


Nmap run completed -- 1 IP address (1 host up) scanned in 1.440 seconds
```



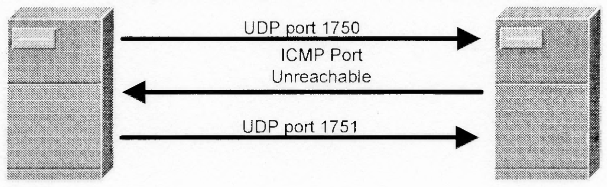
```
Frame 1 (54 bytes on wire, 54 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.120 (192.168.1.120), Dst Addr: 130.156.1.15 (130.156.1.15)
Transmission Control Protocol, Src Port: 39289 (39289), Dst Port: http (80), Seq: 0, Ack: 0, Len: 0
   Flags: 0x0029 (FIN, PSH, URG)

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
20010  00 28 de 99 00 00 30 06 66 6b c0 a8 01 78 82 9c   .(....0.fk...x..
0020  01 0f 99 79 00 50 66 ed 38 fe 00 00 00 00 50 29   ...y.Pf.8.....P)
0030  04 00 2c 3b 00 00                                 ..,;..

Frame 2 (54 bytes on wire, 54 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.120 (192.168.1.120), Dst Addr: 130.156.1.15 (130.156.1.15)
Transmission Control Protocol, Src Port: 39289 (39289), Dst Port: 27 (27), Seq: 0, Ack: 0, Len: 0
   Flags: 0x0029 (FIN, PSH, URG)

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 28 1b 1d 00 00 27 06 32 e8 c0 a8 01 78 82 9c   .(....'.2....x..
0020  01 0f 99 79 00 1b 66 ed 38 fe 00 00 00 00 50 29   ...y..f.8.....P)
0030  10 00 20 70 00 00                                 .. p..

Frame 3 (60 bytes on wire, 60 bytes captured)
Ethernet II, Src: 00:06:25:80:db:93, Dst: 00:b0:d0:05:43:ec
Internet Protocol, Src Addr: 130.156.1.15 (130.156.1.15), Dst Addr: 192.168.1.120 (192.168.1.120)
Transmission Control Protocol, Src Port: 27 (27), Dst Port: 39289 (39289), Seq: 0, Ack: 0, Len: 0
   Flags: 0x0014 (RST, ACK)

0000  00 b0 d0 05 43 ec 00 06 25 80 db 93 08 00 45 00   ....C...%.....E.
0010  00 28 37 92 00 00 30 06 0d 73 82 9c 01 0f c0 a8   .(7...0..s......
0020  01 78 00 1b 99 79 00 00 00 00 66 ed 38 fe 50 14   .x...y....f.8.P.
0030  00 00 30 85 00 00 00 00 00 00 00 00               ..0.........

Frame 4 (54 bytes on wire, 54 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.120 (192.168.1.120), Dst Addr: 130.156.1.15 (130.156.1.15)
Transmission Control Protocol, Src Port: 39290 (39290), Dst Port: http (80), Seq: 0, Ack: 0, Len: 0
   Flags: 0x0029 (FIN, PSH, URG)

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 28 3c 31 00 00 32 06 06 d4 c0 a8 01 78 82 9c   .(<1..2.......x..
0020  01 0f 99 7a 00 50 66 ec 38 ff 00 00 00 00 50 29   ...z.Pf.8.....P)
0030  0c 00 24 3a 00 00                                 ..$:..
```

Figure 2.14 TCP XMAS connect.

2.3.1.11 UDP Connect. UDP scanning is not as reliable as TCP scanning. Because there is no handshake process, a listening UDP port will not generate a response. Most invalid UDP ports will generae an ICMP port unreachable as seen below when a scan is generated using nmap.

It is impossible to distinguish between an open UDP port on a scanned server and a port that has been blocked by a firewall, this explains the resulting output of the scan displayed below. Port 7051 in this case is open and 7050 is closed on the scanned server.

```
nmap -sU 130.156.1.41 -p 7050,7051 -v -P0 -n

Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at 2004-09-27 21:34 EDT Initiating UDP Scan against 130.156.1.41 [2 ports] at 21:34 The UDP
Scan took 1.29s to scan 2 total ports.
Host 130.156.1.41 appears to be up ... good.
Interesting ports on 130.156.1.41:
PORT     STATE         SERVICE
7050/udp closed        unknown
7051/udp open|filtered unknown
```



```
Nmap run completed -- 1 IP address (1 host up) scanned in 1.438 seconds
```

```
Frame 1 (42 bytes on wire, 42 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.123 (192.168.1.123), Dst Addr: 130.156.1.41 (130.156.1.41)
User Datagram Protocol, Src Port: 44236 (44236), Dst Port: 7051 (7051)

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 1c 71 ff 00 00 27 11 db e9 c0 a8 01 7b 82 9c   ..q...'......{..
0020  01 29 ac cc 1b 8b 00 08 f1 9d                     .)........

Frame 2 (42 bytes on wire, 42 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.123 (192.168.1.123), Dst Addr: 130.156.1.41 (130.156.1.41)
User Datagram Protocol, Src Port: 44236 (44236), Dst Port: 7050 (7050)

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 1c 90 68 00 00 26 11 be 80 c0 a8 01 7b 82 9c   ...h..&......{..
0020  01 29 ac cc 1b 8a 00 08 f1 9e                     .)........

Frame 3 (70 bytes on wire, 70 bytes captured)
Ethernet II, Src: 00:06:25:80:db:93, Dst: 00:b0:d0:05:43:ec
Internet Protocol, Src Addr: 130.156.1.41 (130.156.1.41), Dst Addr: 192.168.1.123 (192.168.1.123)
Internet Control Message Protocol
User Datagram Protocol, Src Port: 44236 (44236), Dst Port: 7050 (7050)

0000  00 b0 d0 05 43 ec 00 06 25 80 db 93 08 00 45 00   ....C...%.....E.
0010  00 38 9e c5 00 00 73 01 63 17 82 9c 01 29 c0 a8   .8....s.c....)..
0020  01 7b 03 03 44 df 00 00 00 00 45 00 00 1c 90 68   .{..D.....E....h
0030  00 00 19 11 7c 9e c0 a8 01 7b 82 9c 01 29 ac cc   ....|....{...)..
0040  1b 8a 00 08 3e a1                                 ....>.

Frame 4 (42 bytes on wire, 42 bytes captured)
Ethernet II, Src: 00:b0:d0:05:43:ec, Dst: 00:06:25:80:db:93
Internet Protocol, Src Addr: 192.168.1.123 (192.168.1.123), Dst Addr: 130.156.1.41 (130.156.1.41)
User Datagram Protocol, Src Port: 44237 (44237), Dst Port: 7051 (7051)

0000  00 06 25 80 db 93 00 b0 d0 05 43 ec 08 00 45 00   ..%.......C...E.
0010  00 1c a5 aa 00 00 27 11 a8 3e c0 a8 01 7b 82 9c   ......'..>...{..
0020  01 29 ac cd 1b 8b 00 08 f1 9c                     .)........
```

Figure 2.15 UDP connect.

2.3.2 Scan and Probe Tools

NMAP

Nmap ("Network Mapper") is an extremely popular open source utility for network exploration and security auditing. Its most fundamental use if to perform basic network scanning. Its flexile design allows it to easily scan a single host or a set of large networks. It is capable of performing both aggressive wide scans as well as stealthy targeted scans. Through a number of techniques, nmap is able to determine what hosts are available on the network, what services those hosts are offering, what operating systems they are running, what type of packet filters/firewalls are in use, and more.

Nmap supports a large number of scanning techniques such as: UDP, TCP connect(), TCP SYN (half open), ftp proxy (bounce attack), ICMP (ping sweep), FIN, ACK sweep, Xmas Tree, SYN sweep, IP Protocol, and Null scan. Nmap also offers a number of advanced features such as remote OS detection via TCP/IP fingerprinting, stealth scanning, dynamic delay and retransmission calculations, parallel scanning, detection of down hosts via parallel pings, decoy scanning, port filtering detection, direct (non-portmapper) RPC scanning, fragmentation scanning, and flexible target and port specification. [36]

HPING

Inspired by the ping unix command, hping is a simple to use tool that allows for a great deal of flexibility in creating, analyzing, and sending network packets. hping is a command-line oriented TCP/IP packet assembler/analyzer that supports TCP, UDP, ICMP and RAW-IP protocols. While hping was mainly used as a security tool in the past, it can be used in many ways to test networks and hosts. Some of the fundamental features of hping include [37]:

- Advanced port scanning

- Network testing, using different protocols, TOS, fragmentation
- Advanced traceroute, under all the supported protocols
- Remote OS fingerprinting
- TCP/IP stacks auditing
- Rate control of generated packets

Rich set of packet generation options

2.3.3 Scan and Probe Tests Performed

Detection of a scan and probe attack must take into consideration the source(s), destination(s), source port(s), and destination ports(s) in order to verify that a scan or probe is occurring. By design, the RAP algorithm of the Conex IDS system takes into consideration how frequently these parameters appeared during system training.

In order to accurately perform testing, a number of scenarios needed to be tested for each type of scan and probe attack. The following eight variations of testing occurred for each scan and probe attack type.

The first four test types utilized connection attempts from a single source address/port pair to a single destination address/port pair.

- Test 1 consists of a common IP address and common port. This test utilizes IP address and port combinations that were utilized during training, and are represented in the Services.list file
- Test 2 consists of common IP address and uncommon port. This test utilizes IP addresses that was utilized during training, and represented in the Services.list file. Ports used in this test were not utilized during training and are not present in the services.list file
- Test 3 consists of uncommon IP address and common port. This test utilizes ports frequently utilized during training and present in the services.list file. The

IP addresses used in this test were not utilized during training and were not represented in the services.list file.

- Test 4 consists of uncommon IP address and uncommon port. This test utilizes only IP address and port combinations that were not utilized during training and are not present in the services.list file

The next set of tests utilized multiple connection attempts with varying address or port destinations, creating a series of host sweeps and port sweeps.

- Test 5 consists of a set of common IP address and common ports. This test utilizes IP address and port combinations that were utilized during training, and are represented in the Services.list file.
    - Part A consists of a host sweep
    - Part B consists of a port sweep
- Test 6 consists of a set of common IP address and uncommon ports. This test utilizes IP addresses that was utilized during training, and represented in the Services.list file. Ports used in this test were not utilized during training and are not present in the services.list file
    - Part A consists of a host sweep
    - Part B consists of a port sweep
- Test 7 consists of a set of uncommon IP address and common ports. This test utilizes ports frequently utilized during training and present in the services.list file. The IP addresses used in this test were not utilized during training and were not represented in the services.list file.
    - Part A consists of a host sweep
    - Part B consists of a port sweep
- Test 8 consists of a set of set of uncommon IP address and uncommon ports. This test utilizes only IP address and port combinations that were not utilized during training and are not present in the services.list file.
    - Part A consists of a host sweep
    - Part B consists of a port sweep

2.3.4 Scan and Probe Test Results

Initial scan and probe tests were inconclusive. The algorithms used in testing were found to be very dependant on the training results, leading to an alert from any packet with an uncommon port destination. This lead to very poor results in the specific environment where testing was performed. The introduction of the RAC correlator allowed these alerts to be dropped and led to better results. Additionaly, ICMP and UDP protocols were not working properly during initial testing.

Once UDP and ICMP protocols were functional and the RAC algorithm was used to reduce the number of false positives seen, results were much more positive. Results of scan and probe testing are summarized in Table 2.3 and Table 2.4

Table 2.3 Scan and Probe Results – Initial Test

Scan Type	1	2	3	4	5a	5b	6a	6b	7a	7b	8a	8b
ICMP Echo - IDS	×	X	×	X	×		X		×		X	
ICMP Echo – TP	×	X	×	X	×		X		×		X	
ICMP Timestamp – IDS	×	X	×	X	×		X		×		X	
ICMP Timestamp – TP	×	X	×	X	×		X		×		X	
ICMP Mask – IDS	×	X	×	X	×		X		×		X	
ICMP Mask – TP	×	X	×	X	×		X		×		X	
ICMP Info Request – IDS	×	X	×	X	×		X		×		X	
ICMP Info Request – TP	×	X	×	X	×		X		×		X	
UDP Scan – IDS	×	×	×	×	×	×	×	×	×	×	×	×
UDP Scan – TP	×	×	×	×	×	×	✓	✓	✓	✓	✓	✓
TCP Xmas Scan – IDS	×	✓	✓	✓	×	×	✓	✓	✓	✓	✓	✓
TCP Xmas Scan – TP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
TCP Null Scan – IDS	×	✓	✓	✓	×	×	✓	✓	✓	✓	✓	✓
TCP Null Scan – TP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
TCP FIN scan – IDS	×	✓	✓	✓	×	×	✓	✓	✓	✓	✓	✓
TCP FIN scan – TP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SYN Stealth Scan – IDS	×	✓	✓	✓	×	×	×	×	×	×	×	×
SYN Stealth Scan – TP	×	✓	✓	✓	×	×	×	×	×	×	×	×
Connect Scan – IDS	×	✓	✓	✓	×	×	✓	✓	✓	✓	✓	✓
Connect Scan – TP	×	×	×	×	×	×	×	×	×	×	×	×
TCP Connect - IDS	×	✓	✓	✓	×	×	✓	✓	✓	✓	✓	✓
TCP Connect – TP	×	×	×	×	×	×	×	×	×	×	×	×

Table 2.4 Scan and Probe Results – Final Test

Scan Type	1	2	3	4	5a	5b	6a	6b	7a	7b	8a	8b
ICMP Echo - IDS	✗	X	✗	X	✓		X		✓		X	
ICMP Echo – TP	✗	X	✗	X	✗		X		✗		X	
ICMP Timestamp – IDS	✓	X	✓	X	✓		X		✓		X	
ICMP Timestamp – TP	✗	X	✗	X	✗		X		✗		X	
ICMP Mask – IDS	✓	X	✓	X	✓		X		✓		X	
ICMP Mask – TP	✗	X	✗	X	✗		X		✗		X	
ICMP Info Request – IDS	✗	X	✗	X	✓		X		✓		X	
ICMP Info Request – TP	✗	X	✗	X	✗		X		✗		X	
UDP Scan – IDS	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗	✓
UDP Scan – TP	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓
TCP Xmas Scan – IDS	✗	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓
TCP Xmas Scan – TP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
TCP Null Scan – IDS	✗	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓
TCP Null Scan – TP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
TCP FIN scan – IDS	✗	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓
TCP FIN scan – TP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SYN Stealth Scan – IDS	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗
SYN Stealth Scan – TP	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗
Connect Scan – IDS	✗	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓
Connect Scan – TP	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
TCP Connect - IDS	✗	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓
TCP Connect – TP	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗

2.4 Results

Although a good deal of effort was needed to increase the accuracy and usability of the Conex IDS product, final stages of testing are positive. Typical results of scan and probe testing reveal miscalculation rates of less than .064%, false positive rates as low as .074%, and false negative rates of less than .012%. Denial of service attack testing reveals miscalculation rates of .08%, False positive rates of .09%, and a false negative rate of 0%.

Manually matching the results of Conex DOS detection to an export of the college's Tipping Point IPS reveals the following results

Detecting known DOS attacks

- Source = truncated dump file @ dump_sources[0] '../../2-17-05dump'
    - Number of observed samples = 4107
    - Number of positive detections = 75
    - Number of negative detections = 4032
    - Number of false negative detections = 0
    - Number of false positive detections = 7
- Source = truncated dump file @ dump_sources[0] '../../feb15bergendump'
    - Number of observed samples = 1004
    - Number of positive detections = 10
    - Number of negative detections = 994

- Number of false positive=3
- Number of false negative=1 (1.1.1.50)

CHAPTER 3

INTRUSION DETECTION ENHANCEMENTS

3.1 Communication Mechanisms

Communication is a key component to the design of the ongoing security project. In order to take advantage of the various security detection technologies working throughout the different modules of the project, a flexible and scalable communication mechanism must be included. The intersite communication module is this component.

As a standalone solution, the current security project is very modular and capable of existing autonomously. Its ability to provide security alerts would be limited primarily by its ability to capture all network traffic necessary to detect modern network security alert events.

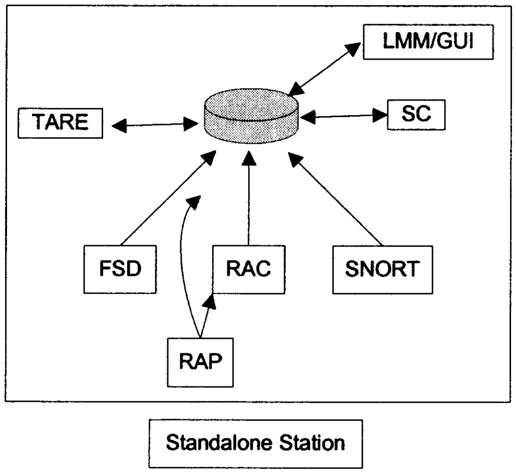


Figure 3.1 Standalone Conex station.

The intersite communication module is to be included in each security product, enabling the ability to share detected information with other devices in order to fully leverage the detection capabilities of the related technologies. Each security station is designed to maintain an independent database containing all alerts originating from any of the detection mechanisms available to that station. The intersite communication module allows each device to add security alerts generated from other devices in order to maximize the detection capabilities of all security stations. Correlators can examine relevant security alerts from multiple locations in order to gain a more complete perspective of security relevant events throughout a network of any size.

To ensure flexibility in the creation and design of a security alert, Intrusion Detection Message Exchange Format [38] was adopted into the basic intersite communication design. The top-level class for all IDMEF messages is IDMEF-Message. There are two types of messages defined; Alerts and Heartbeats. Subclasses of the message class are used to provide the detailed information carried in the message.

The IDMEF data model dictates how an alert should be formatted. The interperetation of the data contained within a message is left to the specific implementation of the defined format. The basic subclasses of an IDMEF message include the following.

- **Analyzer** - Identification information for the analyzer that originated the alert.
- **CreateTime** - The time the alert was created. Of the three times that may be provided with an Alert, this is the only one that is required.
- **Classification** - The "name" of the alert, or other information allowing the manager to determine what it is.

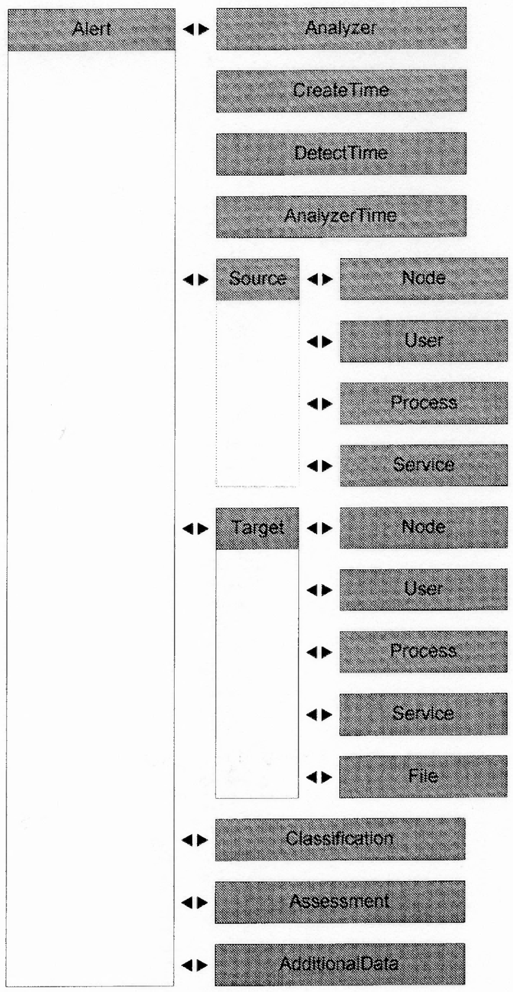


Figure 3.2 General IDMEF message.

- DetectTime - The time the event(s) leading up to the alert was detected. In the case of more than one event, the time the first event was detected. In some circumstances, this may not be the same value as CreateTime.

- AnalyzerTime - The current time on the analyzer

- Source - The source(s) of the event(s) leading up to the alert.

- Target - The target(s) of the event(s) leading up to the alert.

- Assessment - Information about the impact of the event, actions taken by the analyzer in response to it, and the analyzer's confidence in its evaluation.

- AdditionalData - Information included by the analyzer that does not fit into the data model. This may be an atomic piece of data, or a large amount of data provided through an extension to the IDMEF

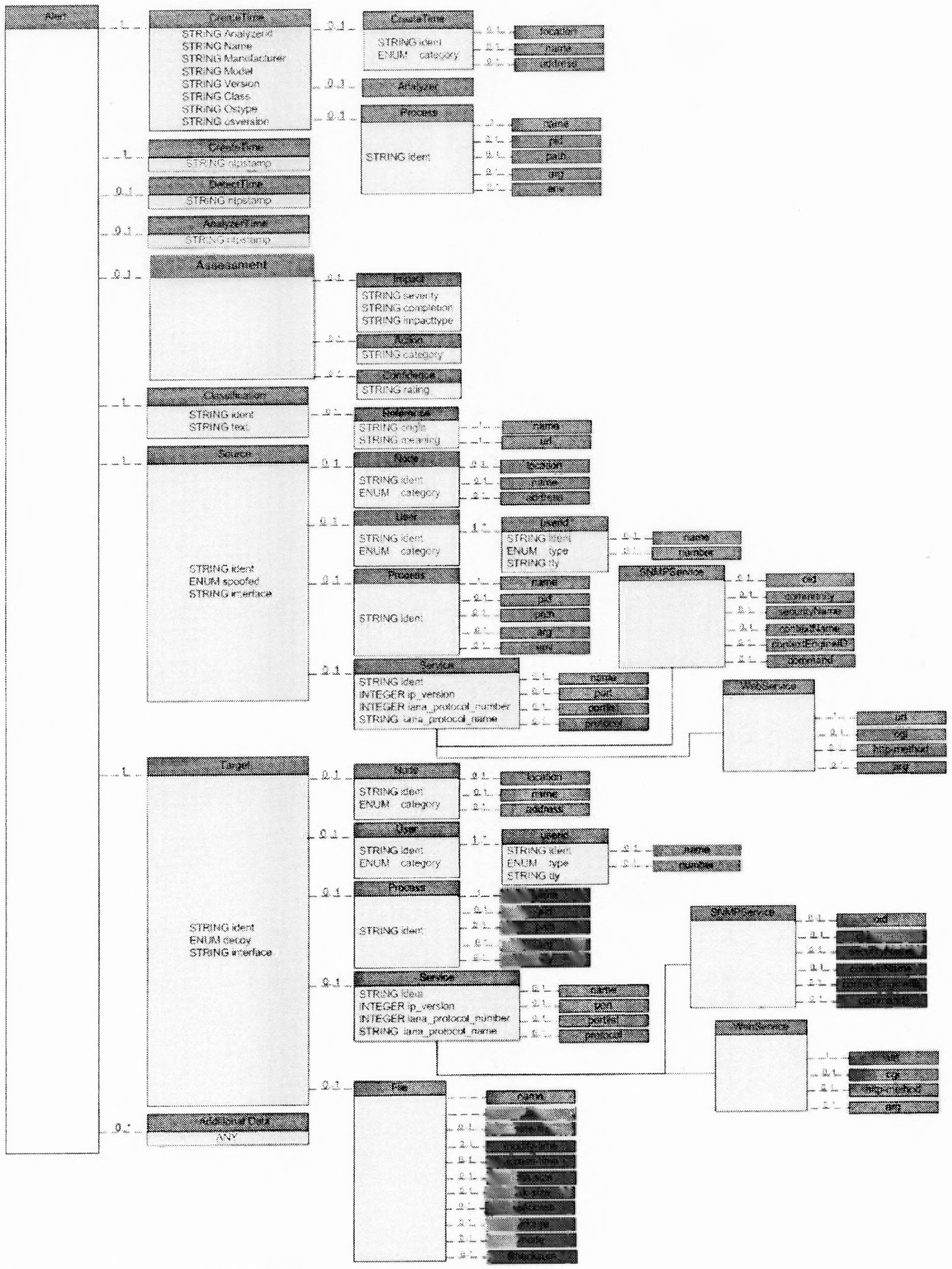


Figure 3.3 Full IDMEF message.

In order to best utilize the standard, relevant portions of the full IDMEF standard were identified in order to create the basic structure of an IDMEF alert to be used by our security project.

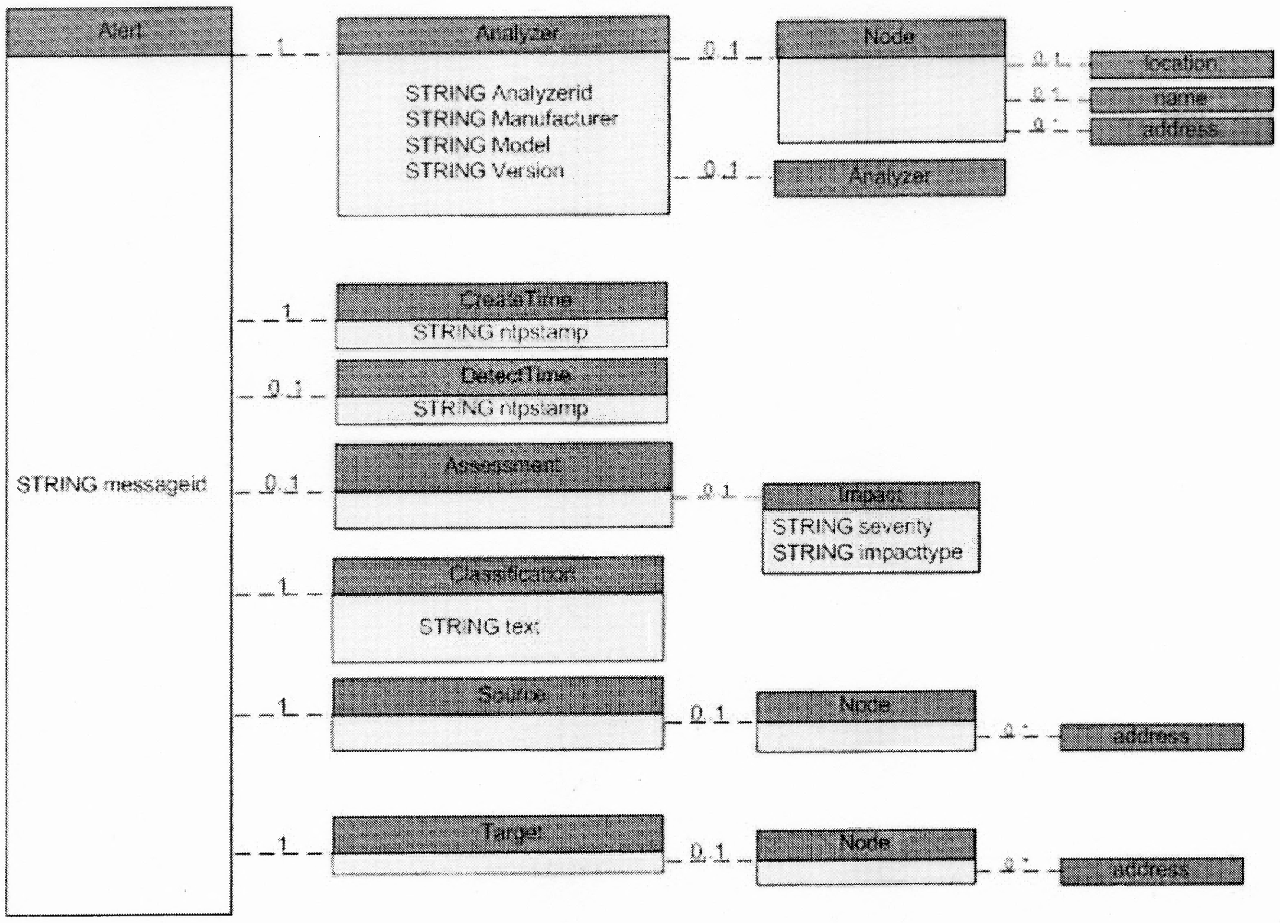


Figure 3.4 Modified IDMEF message.

A description of the fields that make up the designed IDMEF message can be found in the following table.

Table 3.1 Modified IDMEF message

FIELD	TYPE	DESCRIPTION
Messageid	Primary Key	A unique identifier for the alert
CreateTime	Time	NTP timestamps MUST be encoded as two 32-bit hexadecimal values, separated by a period ('.'). For example, "0x12345678.0x87654321".
DetectTime	Time	This will be the same as the above timestamp in most cases, but RAC may be able to provide a separate create and detect timestamp in the future.
Text	String	The alert message
SourceIP	String	IP address of the attacking node
TargetIP	String	IP address of the Victim Node
SiteId	String	This field will repeat the information found in SYSinfo, but may simplify the initial coding of the Intersite communication module.
AlertModule	String	This field will identify the module from which the alert information was received. While it could be derived from the Text of the alert, a separate field will substantially simplify programming the intersite module.
Analyzerid	STRING	A unique identifier for the analyzer manufacturer We can use the hostname (BCCConex.bergen.edu)
Model	String	The model name/number of the analyzer software and/or hardware. We would create a model name for the IDS (I used 100S1 in the example for 100 Megabit Sensor v1)
version	String	The version number of the analyzer software and/or hardware I used 1.0
location	String	The location of the equipment. I used DMZ Network
Address	String	The network or hardware address of the equipment. I used the IP 172.18.0.54
Name	String	The name of the equipment We can use the first part of the hostname (BCCCONEX)
Manufacturer	String	The equipment Manufacturer I used CONEX
SrcPort	Number	The source port of the alert
SrcProtocol	String	The source Protocol (ICMP,UDP,TCP)
DstPort	Number	The destination port of the alert
DstProtocol	String	The Destination Protocol (ICMP,UDP,TCP)

The output of RAP and FSD modules are identical. The MySQL fields generated by a RAP and FSD alert include:

- TimeStamp
- SrcIP
- SrcPort
- DstIP
- DstPort
- AttackType
- Protocol

Due to the lack of a unique identifier in the output of RAP and FSD, this identifier must be created when the alert is reported to the intersite communication module.

IDMEF defines two types of messages; Alerts and Heartbeats. FSD and RAP output will fit easily into the IDMEF alert message format as shown below.

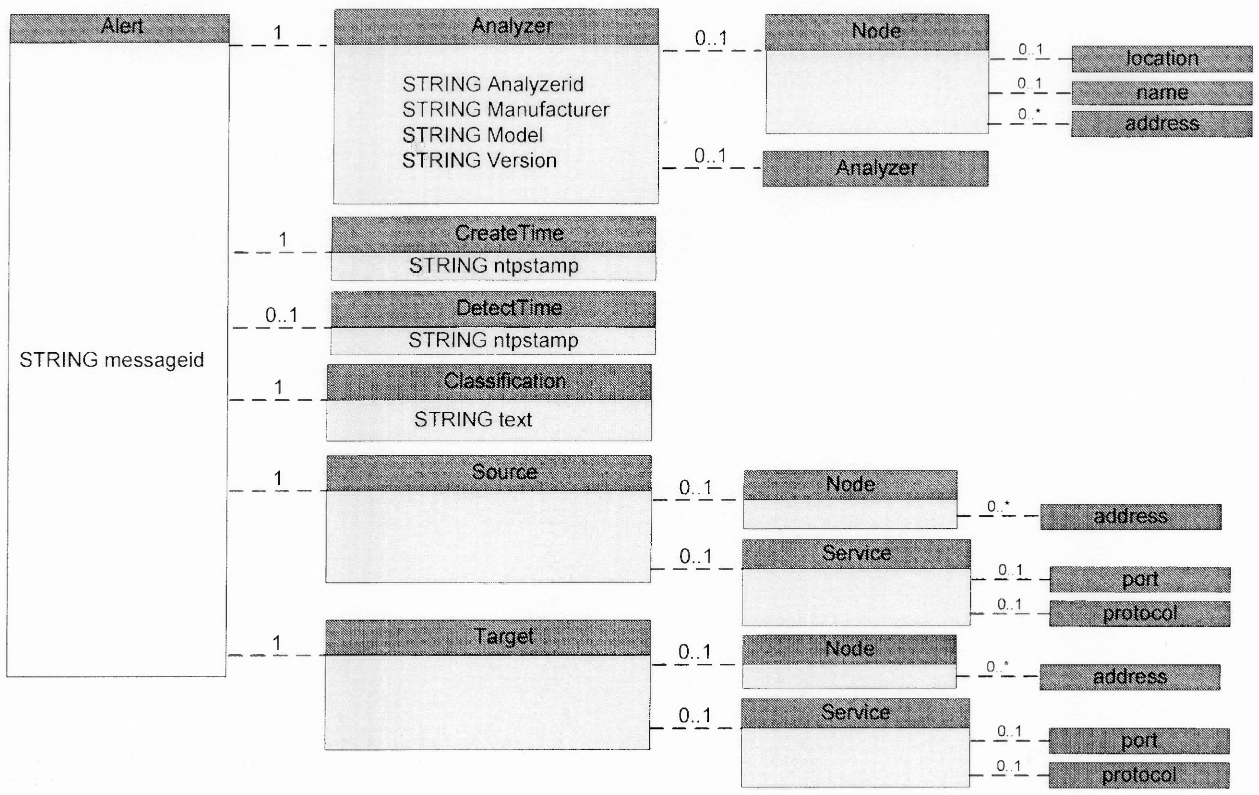


Figure 3.5 Sample RAP or FSD message.

A sample RAP or FSD alert formatted into an IDMEF message to be used in the intersite communication module is shown below.

```
     <?xml version="1.0" encoding="UTF-8" ?>
 - <idmef:IDMEF-Message xmlns:idmef="http://iana.org/idmef" version="1.0">
 - <idmef:Alert messageid="0000000001">
 - <idmef:Analyzer analyzerid="BCCCONEX.bergen.edu" Manufacturer="CONEX"
       Model="100S1" Version="1.0">
 - <idmef:Node>
   <idmef:location>DMZ Network</idmef:location>
   <idmef:name>BCCCONEX.bergen.edu</idmef:name>
   <idmef:address>172.18.0.54</idmef:address>
     </idmef:Node>
     </idmef:Analyzer>
   <idmef:CreateTime ntpstamp="0xbc723b45.0xef449129">2000-03-
       09T10:01:25.93464-05:00</idmef:CreateTime>
   <idmef:DetectTime ntpstamp="0xbc723b45.0xef449129">2000-03-
       09T10:01:25.93464-05:00</idmef:DetectTime>
 - <idmef:Source>
 - <idmef:Node>
 - <idmef:Address>
   <idmef:address>192.0.2.50</idmef:address>
     </idmef:Address>
     </idmef:Node>
 - <idmef:Service>
 - <idmef:port>
   <idmef:port>6859</idmef:port>
     </idmef:port>
 - <idmef:protocol>
   <idmef:protocol>TCP</idmef:protocol>
     </idmef:protocol>
     </idmef:Service>
     </idmef:Source>
 - <idmef:Target>
 - <idmef:Node>
 - <idmef:Address>
   <idmef:address>172.18.0.35</idmef:address>
     </idmef:Address>
     </idmef:Node>
 - <idmef:Service>
 - <idmef:port>
   <idmef:port>6667</idmef:port>
     </idmef:port>
 - <idmef:protocol>
   <idmef:protocol>TCP</idmef:protocol>
     </idmef:protocol>
     </idmef:Service>
     </idmef:Target>
   <idmef:Classification text="Teardrop detected" ident="00008" />
     </idmef:Alert>
     </idmef:IDMEF-Message>
```

Figure 3.6 Sample RAP/FSD alert.

RAC output defines a set of related RAP alerts instead of individual alerts. As such, neither of the defined IDMEF message types will suit its output. Currently RAP outputs the following fields:

- ScenID
- StartTime
- LastTime
- ServerIP
- ClientIP
- AlertNum
- Randomness

Because this does not adequately fall into either the alert or heartbeat messages defined by the basic IDMEF format, the IDMEF DTD will need to be extended to accommodate the AlertNum and Randomness values.

The use of IDMEF AdditionalData fields will allow a custom IDMEF Summary message type to be created. The new message type is shown below.

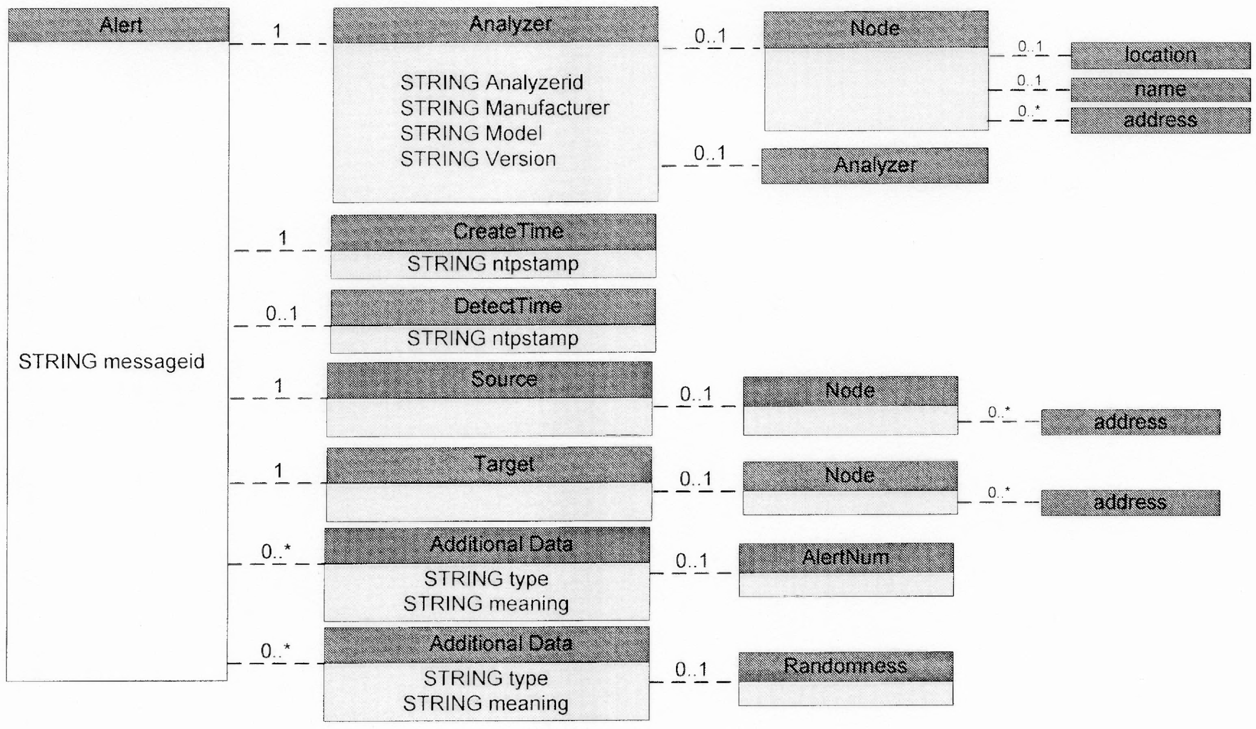


Figure 3.7 Modified RAC message.

A sample RAC message as defined by our new message type is shown below.

```
     <?xml version="1.0" encoding="UTF-8" ?>
 - <idmef:IDMEF-Message xmlns:idmef="http://iana.org/idmef" version="1.0">
 - <idmef:Alert messageid="0000000001">
 - <idmef:Analyzer analyzerid="BCCCONEX.bergen.edu" Manufacturer="CONEX" Model="100S1"
      Version="1.0">
 - <idmef:Node>
   <idmef:location>DMZ Network</idmef:location>
   <idmef:name>BCCCONEX.bergen.edu</idmef:name>
   <idmef:address>172.18.0.54</idmef:address>
     </idmef:Node>
     </idmef:Analyzer>
   <idmef:CreateTime ntpstamp="0xbc723b45.0xef449129">2000-03-09T10:01:25.93464-
      05:00</idmef:CreateTime>
   <idmef:DetectTime ntpstamp="0xbc723b45.0xef449129">2000-03-09T10:01:25.93464-
      05:00</idmef:DetectTime>
 - <idmef:Source>
 - <idmef:Node>
 - <idmef:Address>
   <idmef:address>192.0.2.50</idmef:address>
     </idmef:Address>
     </idmef:Node>
 - <idmef:Service>
 - <idmef:port>
   <idmef:port>6859</idmef:port>
     </idmef:port>
 - <idmef:protocol>
   <idmef:protocol>TCP</idmef:protocol>
     </idmef:protocol>
     </idmef:Service>
     </idmef:Source>
 - <idmef:Target>
 - <idmef:Node>
 - <idmef:Address>
   <idmef:address>172.18.0.35</idmef:address>
     </idmef:Address>
     </idmef:Node>
 - <idmef:Service>
 - <idmef:port>
   <idmef:port>6667</idmef:port>
     </idmef:port>
 - <idmef:protocol>
   <idmef:protocol>TCP</idmef:protocol>
     </idmef:protocol>
     </idmef:Service>
     </idmef:Target>
 - <idmef:AdditionalData type="AlertNum" Meaning="Number of Related Alerts">
   <idmef:AlertNum>.50</idmef:AlertNum>
     </idmef:AdditionalData>
 - <idmef:AdditionalData type="Randomness" Meaning="Believability indicator">
   <idmef:Randomness>.50</idmef:Randomness>
     </idmef:AdditionalData>
     </idmef:Alert>
     </idmef:IDMEF-Message>
```

Figure 3.8 Sample RAC message.

With the basic design of our IDMEF messages in place, the design of the communication mechanism can take on several forms.

OPTION 1 – Centralized Approach

In this scenario, IDS devices would report alerts to a designated management station. The management station can be configured to distribute alerts to other IDS systems under its domain. This group of IDS systems could include all systems or a subset of the systems in is management domain.

In most cases DOS, RAC, and Snort (Correlated) alerts would be reported to the central repository. Distribution of full RAP alerts can be optional.

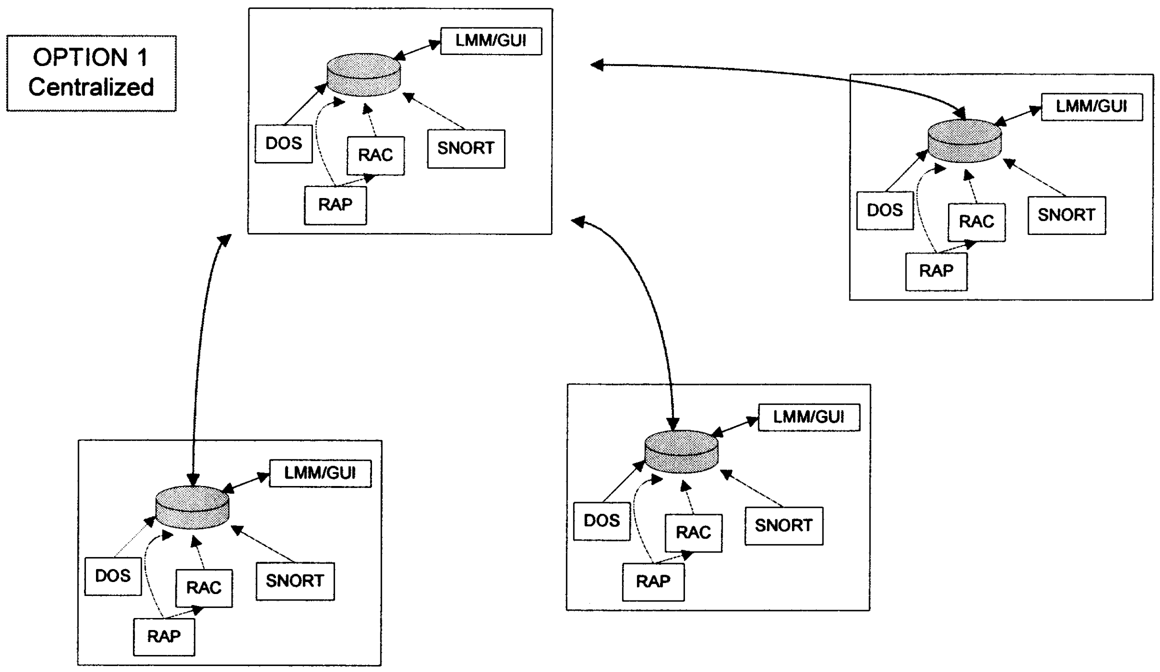


Figure 3.9 Centralized communication module.

The advantages of this approach include:

- Ease of implementation
- Flexible control of information
- Centralized repository of information

Disadvantages include:

- Single point of failure for communication

- Incomplete alert information available to IDS software

OPTION 2 – Distributed Approach

In this scenario, IDS devices under a common management domain would share alert information between them. Each station would have the ability to fully report and analyze all alerts under the management domain.

In most cases DOS, RAC, and Snort (Correlated) alerts would be shared. Distribution of full RAP alerts is likely undesirable in this approach.

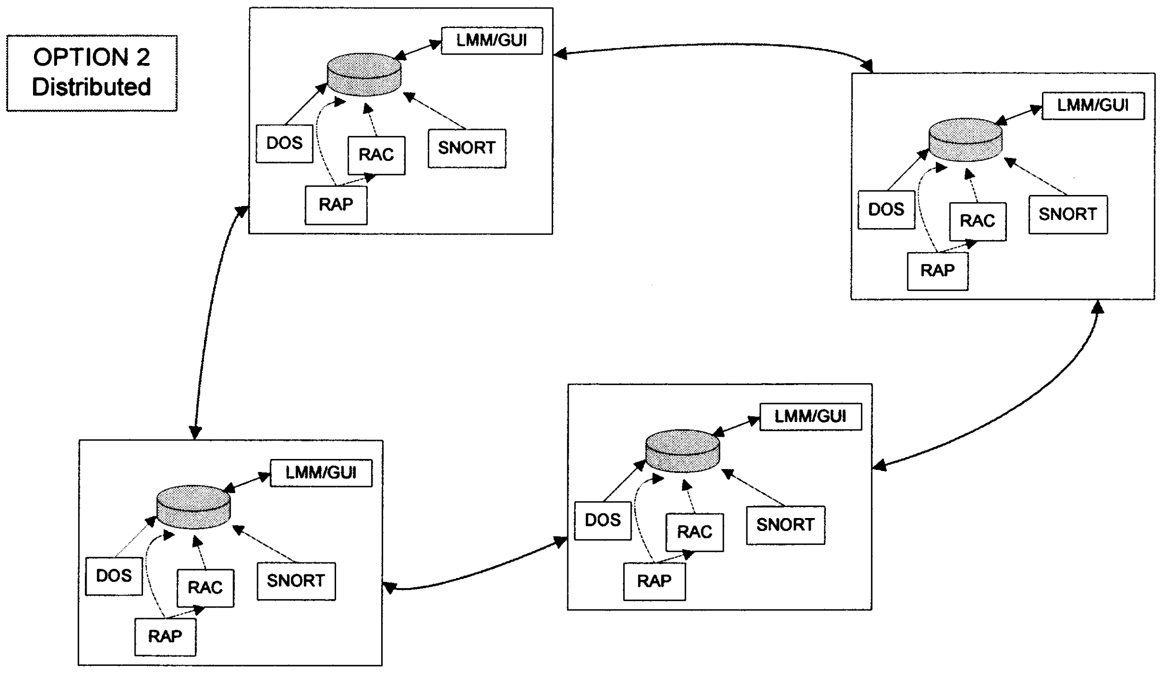


Figure 3.10 Distributed communication module.

The advantages of this approach include:

- Fault tolerance can be easily achieved
- Complete alert information available to each IDS

Disadvantages include:

- Distribution algorithm required

- Complex implementation

While initial implementation of the intersite communication will offer a more basic communication algorithm, the need for a decentralized peer to peer communication mechanism is recognized. To efficiently support communication needs the mechanism must be scalable, efficient, and fault-tolerant. The mechanism for joining and leaving a peer to peer or multicast group must offer authentication. Finally, the mechanism must offer an efficient integration with the functionality of our IDS system in order for the product to benefit from its use.

One possible mechanism for this functionality is Pastry. [41] Offering the basic peer to peer foundation for the desired multicast functionality, it serves as an easily integrated framework for the product's communication requirements.

Pastry [41] creates an efficient mechanism for communicating between participating members through the formation of a communication ring. The communication ring consists of a series of member nodes identified by a unique 128 bit nodeId. The nodeId acts as the principal identifier for communicating nodes. Ring formation, routing decisions, and multicast membership will use the nodeId rather than IP address to function. The nodeIds of participating hosts should be randomly distributed for efficient communication and for purposes of this project can be formed through a simple hash of the member's IP address.

Below is a random sample of 16 addresses and their corresponding communication ring using MD5 as a hash algorithm to ensure proper distribution of nodeIds.

Table 3.2 Communication Ring

IP	MD5 Hash
130.156.1.67	ab9d9b39f6db2847e791c228634e3ab7
130.156.4.12	9ac47c6a1c137255a7c7d27ae69a4364
24.2.3.7	1c41d3f41efe77f6b21577da81dee100
12.58.124.56	ba46d430fb0b6f8558794170aea62009
45.12.36.85	4ab9bed840f91b4eb65e4a287983ccda
45.12.58.69	33eccd9737617b26223e2e15e7fb0a55
45.23.69.4	b090613bf78190e400f4e925be179528
8.4.214.57	695ef4a046411e0a7e267915b4f828ad
7.98.41.24	265002def0423efc56484eaa9914e824
54.56.97.2	1fae728167f827e33cc22058ec52a5b2
21.32.14.48	49648ced6096effa9b6615183bc02df3
87.89.96.5	a141e48d40ee1ebc91724df161cb7c33
25.36.14.8	c9098c2fc8a3166517c60b5063785847
54.74.12.1	f0db359a008f20be81cc2926f928bdc0
65.87.15.214	3f2034414855c0c9412d96e9f1398683
107.25.68.5	025f8fd188903f09dc7fbf68ebf44b85

Once identified with a unique nodeId, each participating node will maintain:

- Leaf set – a set of neighboring clockwise and counterclockwise nodes. The number is dependant on the size of your communication ring
- Routing table – A table of nodeIds arranged in rows identifying nodes with matching prefix's as seen below

Table 3.3 Routing Table

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	1	2	3	4		6			9	a	b	c			f
2	c	6	3	a		9			a	b	a	9			0
5	4	5	e	b		5			c	9	4	0			d
f	1	0	c	9		e			4	d	6	9			b
8	d	0	c	b		f			7	9	d	8			3
f	3	2	d	e		4			c	b	4	c			5
d	f	d	9	d		a			6	3	3	2			9
1	4	e	7	8		0			a	9	0	f			a
8	1	f	3	4		4			1	f	f	c			0
8	e	0	7	0		6			c	6	b	8			0
9	f	4	6	f		4			1	d	0	a			8
0	e	2	1	9		1			3	b	b	3			f
3	7	3	7	1		1			7	2	6	1			2
f	7	e	b	b		e			2	8	f	6			0
0	f	f	2	4		0			5	4	8	6			b
9	6	c	6	e		a			5	7	5	5			e
d	b	5	2	b		7			a	e	5	1			8
c	2	6	2	6		e			7	7	8	7			1
7	1	4	3	5		2			c	9	7	c			c
f	5	8	e	e		6			7	1	9	6			c
b	7	4	2	4		7			d	c	4	0			2
f	7	e	e	a		9			2	2	1	b			9
6	d	a	1	2		1			7	2	7	5			2
8	a	a	5	8		5			a	8	0	0			6
e	8	9	e	7		b			e	6	a	6			f
b	1	9	7	9		4			6	3	e	3			9
f	d	1	f	8		f			9	4	a	7			2
4	e	4	b	3		8			a	e	6	8			8
4	e	e	0	c		2			4	3	2	5			b
b	1	8	a	c		8			3	a	0	8			d
8	0	2	5	d		a			6	b	0	4			c
5	0	4	5	a		d			4	7		7			0

Each entry in the routing table will map a nodeId to the corresponding IP address of the node. The number of total entries in a node's routing table is:

Table 3.4 Table Entries
Source: [43]

Formula	**Example**
$[2^b - 1] * [\log(2^b) N] + 1$ entries or $15 * [\log_{16} N] + 1$	$15 + 1$ 19

The number of rows required for each node can be calculated using the following formulas.

Table 3.5 Table Row Calculations
Source: [43]

Formula	**Example**
$[\log (2^b) N] + l$ or $[\log_{16} N] + l$ entries	5

l is an even integer parameter identifying the total number of neighboring nodes that will be entered in the routing table based solely on proximity in the communication ring.

It is important to realize that the routing table is not a complete table, but instead offers routing capabilities based on nodeId prefixes, offering a higher number of entries for nodes with closely matching nodeIds, enabling each node to efficiently route to more closely matching nodes that can better route the message towards its destination.

Each time a node forwards a message towards its destination, it will route to an entry in its routing table whose nodeId shares a prefix that is at least one digit longer than the current nodeId.

The example below outlines a possible series of routing steps used to send a message originating from node **b090613bf78190e400f4e925be179528** and terminating at node **33eccd9737617b26223e2e15e7fb0a55.**

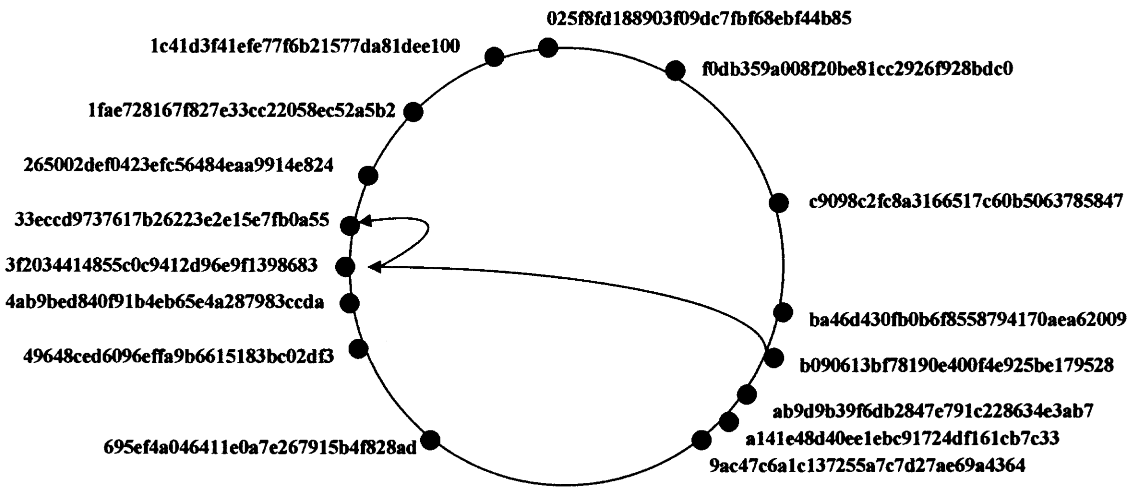

025f8fd188903f09dc7fbf68ebf44b85
1c41d3f41efe77f6b21577da81dee100
f0db359a008f20be81cc2926f928bdc0
1fae728167f827e33cc22058ec52a5b2
265002def0423efc56484eaa9914e824
c9098c2fc8a3166517c60b5063785847
33eccd9737617b26223e2e15e7fb0a55
3f2034414855c0c9412d96e9f1398683
4ab9bed840f91b4eb65e4a287983ccda
ba46d430fb0b6f8558794170aea62009
49648ced6096effa9b6615183bc02df3
b090613bf78190e400f4e925be179528
ab9d9b39f6db2847e791c228634e3ab7
695ef4a046411e0a7e267915b4f828ad
a141e48d40ee1ebc91724df161cb7c33
9ac47c6a1c137255a7c7d27ae69a4364

Figure 3.9 Scribe communication.

SCRIBE

Scribe [42] provides an application level multicast infrastructure built on a peer to peer overlay network such as Pastry. Implemented over Pastry, Scribe offers scalability from one to millions of hosts. Extending the functionality of Pastry to include dynamic management of group members, Scribe offers decentralized creation of multicast groups.

Multicast groups must first be identified by a groupId. To follow this existing example, I will take the MD5 hash of a textual group name. A group is created when an active pastry node requests a JOIN message be delivered using the groupId as a key. Pastry will route the message to the node whose nodeId is closest to the key, designating that node as the root of the group's tree.

Once the group is created and a root is designated, Pastry nodes participate in the multicast group by sending a JOIN message with a destination key of the groupId. As the message is routed towards the root each routing node checks its own membership to the tree specified. If it is a member, it adds the sender as a child of the tree and stops the routing process. If it is not a member, it adds itself to the membership by locally creating an entry for the group, adding the sender as a child, and initiating its own JOIN message for the group.

Table 3.6 Scribe Groups

Group Name	**Hash – (groupId)**
Security Group 1	e2b31eeafc7821749d760b2e5f9b995c

The JOIN process ensures that a single tree is created, allowing existing nodes to add members locally without the need for a central authority on membership. The process of leaving an existing group has similar properties.

If a departing member node has entries in its child table for the group, it will mark itself as no longer a member and continue to route messages up the tree. If a departing node has no children, it will send a LEAVE message to its parent before leaving the group. The parent will remove the entry from its list of children. If the removal results in the parent node having an empty child table, the parent will remove itself from the tree by sending a leave message to its parent.

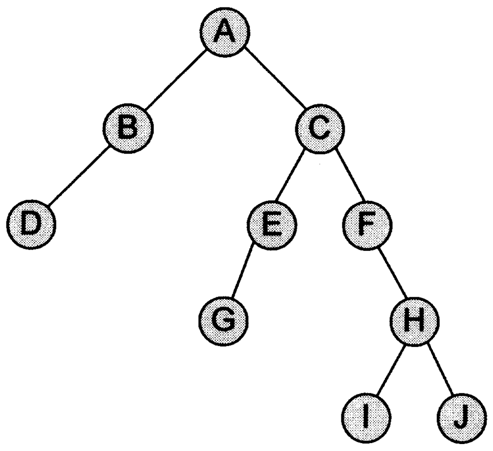
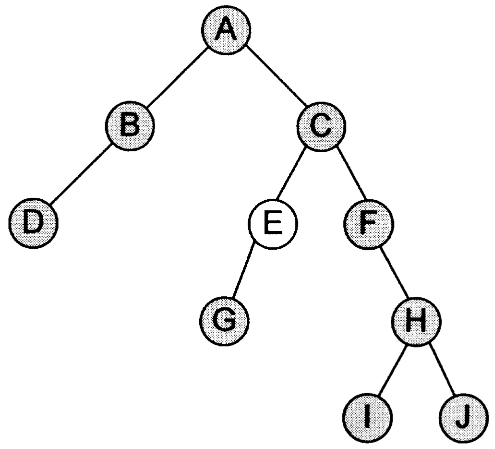


Figure 3.12 Multicast tree – original
Source: [43]



3.13 Multicast tree – after E exits
Source: [43]

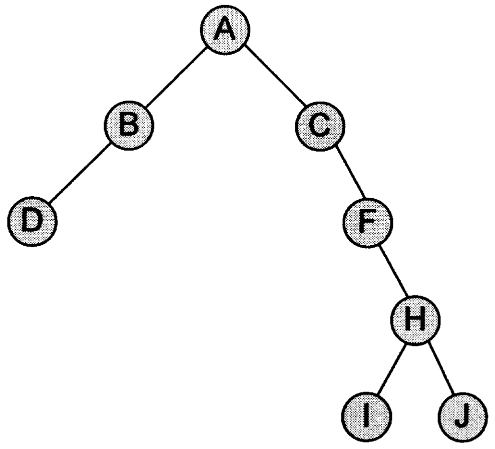


Figure 3.14 Multicast tree – after G exits
Source: [43]

Multicast messages are delivered by addressing the message to the groupId. Once received by the root, the root will forward the message to its children, which in turn will forward the message to their children. Pastry's routing algorithm ensures that multicast group membership trees will be well balanced.

Below is a listing of relevant API entries used to participate in a Pastry network

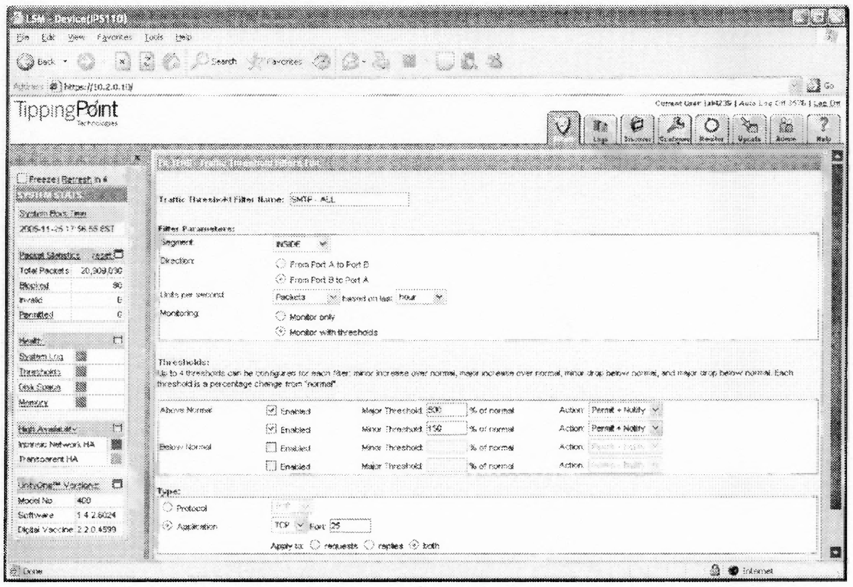
- nodeId = pastryInit(Credentials) – used to join or create a pastry network
- route(msg,key) – routes a message to the node whose nodeId most closely matches the specified key.
- send(msg,IP-addr) – send a message to the specified IP address.

While the Scribe [42] application running on a Pastry [41] foundation can offer a scalable decentralized solution for communication, research [12] indicates that its performance is not effective in small scale solutions. With the growth of peer-to-peer and file-sharing technologies, decentralized communication mechanisms are much more widely adopted and the use of related technologies when combined with a flexible communication format will greatly increase the flexibility of any intrusion detection product.

3.2 Administrator Control

One of the significant challenges in working with an intrusion detection system that relies on anomaly based detection mechanisms is tuning the system to a custom environment. The ability to customize a detection engine to a specific environment can be handled in various ways. In general, the methods can be automatic or administrator controlled. As seen in the description of RIDS, a training mechanism can be used to tune the performance of an anomaly based algorithm. In many cases, however, custom administrator controls can complement even the most sophisticated learning algorithms. Security based products in the industry use a variety of methods to allow for custom tuning of their products in several ways.

Tipping point [6] utilizes traffic threshold filters to enable a simple form of anomaly detection. The UnityOne system keeps track of statistical traffic patterns over time and allows an administrator to create custom thresholds for each type of traffic, each with its own custom action set in response to the detected anomaly


LSM - Device(IPS110)
File Edit View Favorites Tools Help
Back Search Favorites
Address https://10.2.0.10/
TippingPoint Technologies
Logs Discover Configure Monitor Upgrade Admin Help

Freeze Refresh in 4

System Stats

System Boot Time: 2005-11-25 17:56:55 EST

Packet Statistics	reset
Total Packets	20,309,030
Blocked	90
Invalid	6
Permitted	0

Health: System Log, Thresholds, Disk Space, Memory

High Availability: Intrinsic Network HA, Transparent HA

UnityOne™ Versions	
Model No	400
Software	1.4.2.6024
Digital Vaccine	2.2.0.4599

Traffic Threshold Filter Name: SMTP - ALL

Filter Parameters:

Segment: INSIDE

Direction: ( ) From Port A to Port B [✓ From Port B to Port A]

Units per second: Packets based on last: hour

Monitoring: ( ) Monitor only [✓ Monitor with thresholds]

Thresholds:
Up to 4 thresholds can be configured for each filter: minor increase over normal, major increase over normal, minor drop below normal, and major drop below normal. Each threshold is a percentage change from "normal".

Above Normal	☑ Enabled	Major Threshold: 500	% of normal	Action: Permit + Notify
	☑ Enabled	Minor Threshold: 150	% of normal	Action: Permit + Notify
Below Normal	☐ Enabled	Minor Threshold:	% of normal	Action:
	☐ Enabled	Major Threshold:	% of normal	Action:

Type:
( ) Protocol
[✓ Application] TCP Port: 25
Apply to: ( ) requests ( ) replies [✓ both]

Done Internet

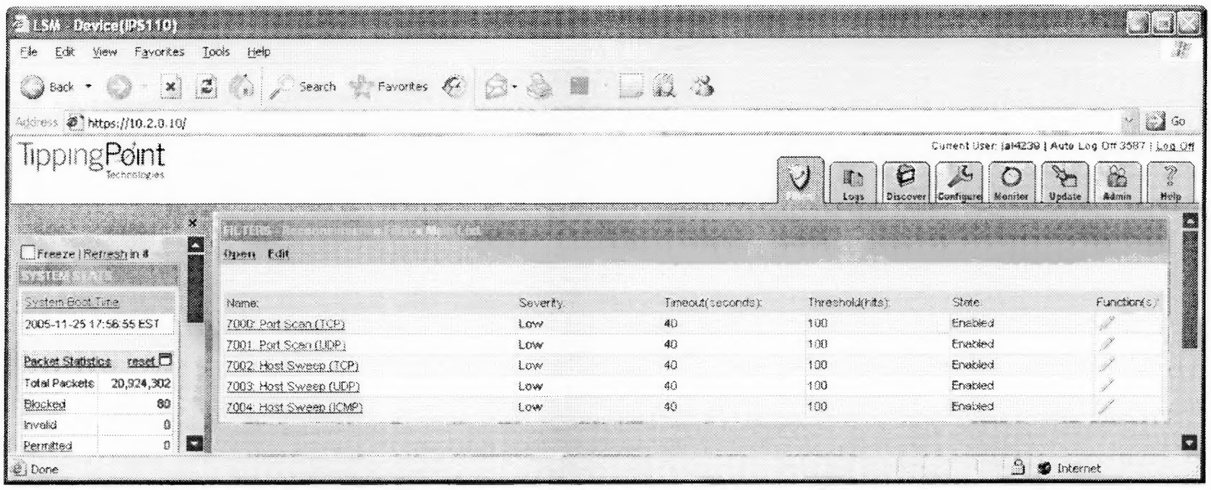
Figure 3.15 Threshold filters.
Source [6]

Tipping Point's UnityOne allows the creation of four types of thresholds for each custom filter:

- minor increase — Traffic is greatly over the set threshold.
- major increase — Traffic is slightly over the set threshold.
- minor decrease — Traffic is slightly below the set threshold.
- major decrease — Traffic is greatly under the set threshold.

In the case above, statistical parameters are based on number of packets, number of bytes, or number of connections. The detected statistical values can be compared to that of normal traffic detected in the past minute, hour, day, 7 days, 30 days, or 35 days. The rich set of statistical values to compare combined with a fairly flexible set of time frames allow for effective tuning by a network administrator.

Similar features in Tipping Point's UnityOne product include custom thresholds for detection of network scanning attempts and SYN flood attacks. The configured parameters for these features include number of hits and number of seconds during which to detect the configured number of hits.

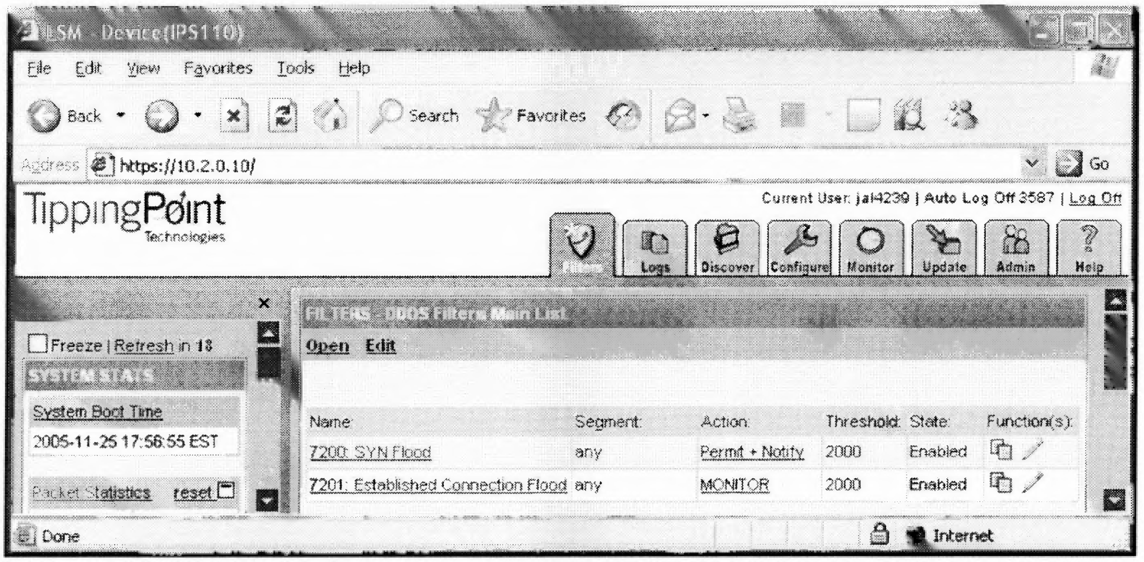

LSM - Device(IPS110)
File Edit View Favorites Tools Help
Back Search Favorites
Address https://10.2.0.10/ Go
TippingPoint Technologies
Current User: jal4239 | Auto Log Off 3587 | Log Off
Logs Discover Configure Monitor Update Admin Help
Freeze | Refresh in 4
SYSTEM STATS
Open Edit
Done Internet

System Boot Time: 2005-11-25 17:56:55 EST

Packet Statistics	reset
Total Packets	20,924,302
Blocked	80
Invalid	0
Permitted	0

Name:	Severity:	Timeout(seconds):	Threshold(hits):	State:	Function(s):
7000: Port Scan (TCP)	Low	40	100	Enabled	
7001: Port Scan (UDP)	Low	40	100	Enabled	
7002: Host Sweep (TCP)	Low	40	100	Enabled	
7003: Host Sweep (UDP)	Low	40	100	Enabled	
7004: Host Sweep (ICMP)	Low	40	100	Enabled	

Figure 3.16 Reconnaissance filters.
Source [6]


LSM - Device(IPS110)
File Edit View Favorites Tools Help
Back Search Favorites
Address https://10.2.0.10/ Go
TippingPoint Technologies
Current User: jal4239 | Auto Log Off 3587 | Log Off
Filters Logs Discover Configure Monitor Update Admin Help
Freeze | Refresh in 18
SYSTEM STATS
FILTERS - DDOS Filters Main List
Open Edit
Done Internet

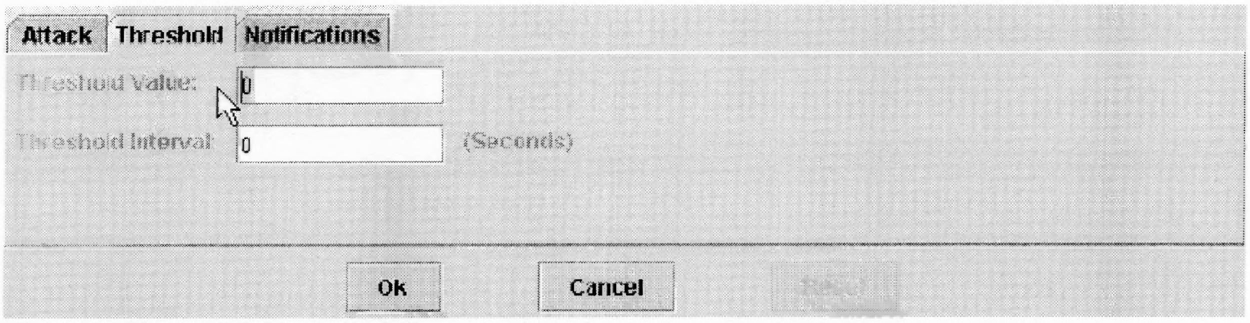
System Boot Time: 2005-11-25 17:56:55 EST

Packet Statistics reset

Name:	Segment:	Action:	Threshold:	State:	Function(s):
7200: SYN Flood	any	Permit + Notify	2000	Enabled	
7201: Established Connection Flood	any	MONITOR	2000	Enabled	

Figure 3.17 SYN flood filters.
Source [6]

McAfee's IntruShield® IPS System [39] supports Threshold Mode, a feature that allows the monitoring and detection of network traffic statistics to detect the presence of traffic threshold anomalies. Configurable parameters for these features include packet count and interval (in seconds).



Threshold Value: 0

Threshold Interval: 0 (Seconds)

Figure 3.18 Threshold dialog box
Source: [39]

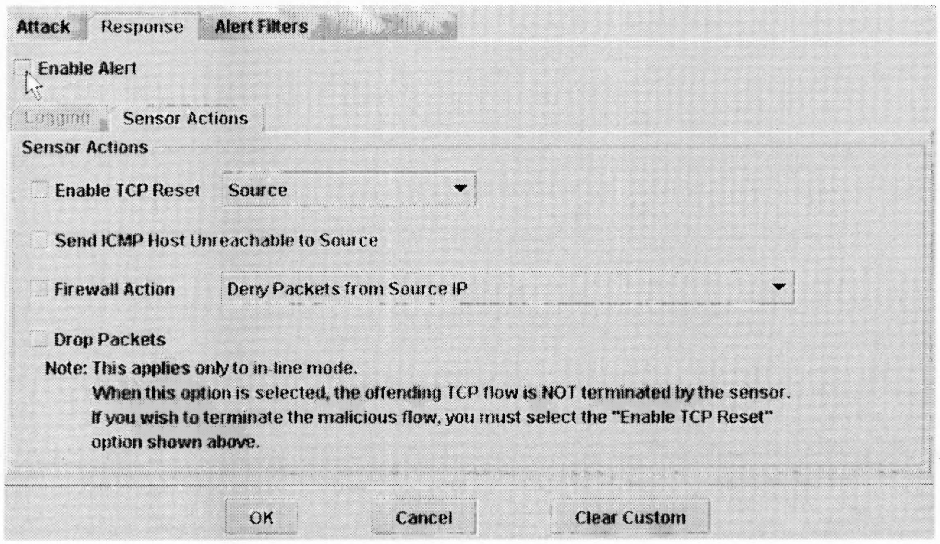
An administrator would need to monitor statistics and provide the threshold information that would work best in the particular environment being monitored. As stated in system documentation, "You must configure the actual thresholds and intervals for each DoS Threshold Mode attack you want to detect. Default thresholds are not provided since the packet rates in every network are different. Customization of DoS thresholds works best after researching the current levels each DoS Threshold attack defends against in order to determine exactly what counts and intervals best protect your network."

Supported threshold mode attacks that can be configured include:

- Link Utilization (Bytes/Sec)
- ICMP Packets
- IP Fragments
- ICMP Packets
- Large UDP Packets
- Rejected UDP Packets
- TCP Connections
- TCP SYNS

- UPD Packets

Each threshold attack is assigned a specific severity and action, allowing administrators to configure a severity level to a higher level correlator and an appropriate response to the custom environment



Enable Alert

Sensor Actions

Enable TCP Reset: Source

Send ICMP Host Unreachable to Source

Firewall Action: Deny Packets from Source IP

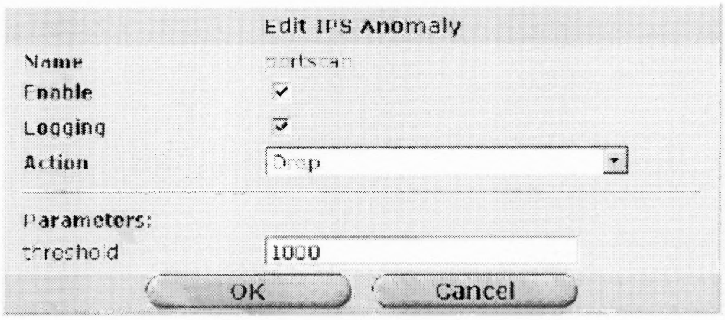
Drop Packets

Note: This applies only to in-line mode.
When this option is selected, the offending TCP flow is NOT terminated by the sensor.
If you wish to terminate the malicious flow, you must select the "Enable TCP Reset" option shown above.

Figure 3.19 Intrusion responses.
Source: [39]

Fortigate's Fortigate-4000 [40] supports administrator controls for anomaly detection. Available administrator controlled anomalies include the following:

- Flooding - If the number of sessions targeting a single destination in one second is over a threshold, the destination is experiencing flooding.
- Scan - If the number of sessions from a single source in one second is over a threshold, the source is scanning.
- Source session limit - If the number of concurrent sessions from a single source is over a threshold, the source session limit is reached.
- Destination session limit - If the number of concurrent sessions to a single destination is over a threshold, the destination session limit is reached.



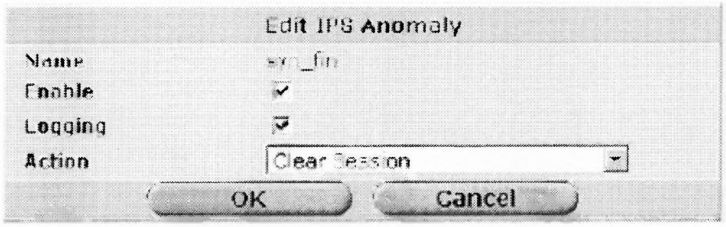
Edit IPS Anomaly

Name	portscan
Enable	✓
Logging	✓
Action	Drop

Parameters:

threshold	1000

Figure 157:Editing the syn_fin IPS anomaly



Edit IPS Anomaly

Name	syn_fin
Enable	✓
Logging	✓
Action	Clear Session

Figure 3.20 Anomaly dialog boxes.
Source: [39]

Custom actions can be configured for each anomaly detected. Available actions include Pass, Drop, Reset, Reset Client, Reset Server, Drop Session, Clear Session, or Pass Session.

Recommendations

Administrator controlled thresholds allow an administrator to identify custom characteristics of a specific environment in order to increase the accuracy and effectiveness of the security device.

Administrators require the assistance of an accurate set of statistics from which comparisons to configured parameters can be made. The statistics should be flexible enough to allow an administrator to specify the statistics that most accurately fit a particular environment.

The resulting alarms generated by the administrator controlled thresholds can be used as independent alarms as in Tipping Point's UnityOne or can be used with a corresponding severity level as input into a higher level correlator.

Administrator controlled thresholds can be used to generate alarms indicating:

- Port scans
- Packet flooding
- Fragmentation attacks
- DOS SYN flooding

Because SYN floods, fragmentation attacks and port scanning are effectively handled by RIDS and RAP modules, administrator controlled thresholds can be combined with the following statistics to detect flooding attacks and provide input into a central correlator to assist in improving detection accuracy.

Statistics to be monitored:

- TCP Total Bytes
- TCP Packet Count
- UDP Total Bytes
- UDP Packet Count
- Large UDP Packets
- ICMP Total Bytes
- ICMP Packet Count
- Large ICMP Packets
- Total Packets
- Total Bytes

- TCP SYN Packets

User Interface should consist of two features

- Protocol Thresholds
- Traffic Thresholds

Protocol thresholds allow for the monitoring and alerting of the behavior of individual protocols.

Traffic thresholds allow for the monitoring of network traffic behavior.

Learning mode monitors traffic and calculates all relevant statistics. Monitoring mode enables alerting based on administrator controlled thresholds.

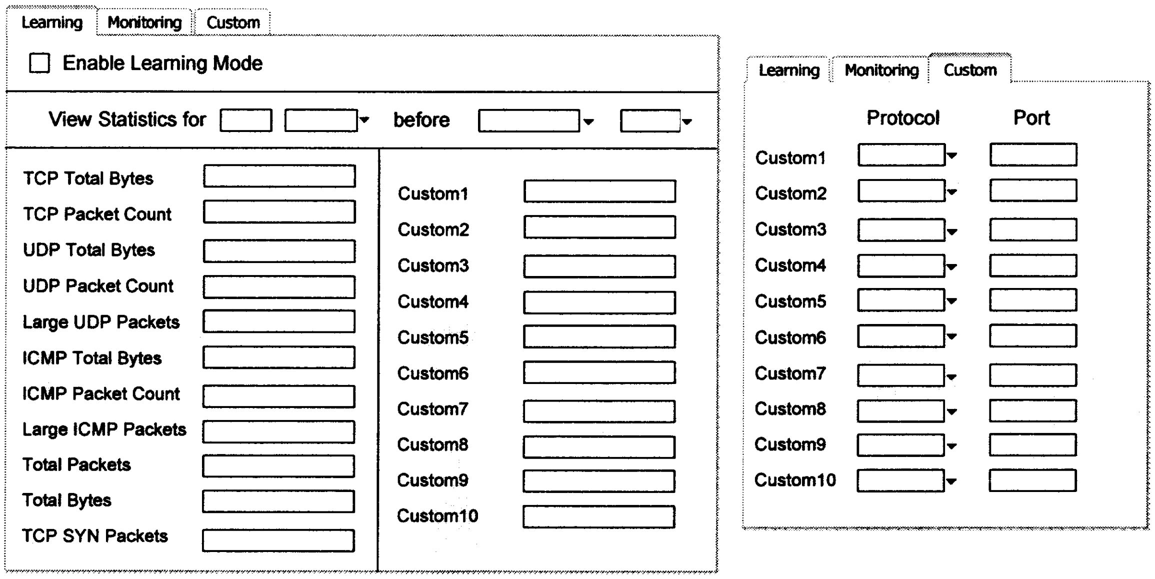


Figure 3.21 Proposed dialog box1.

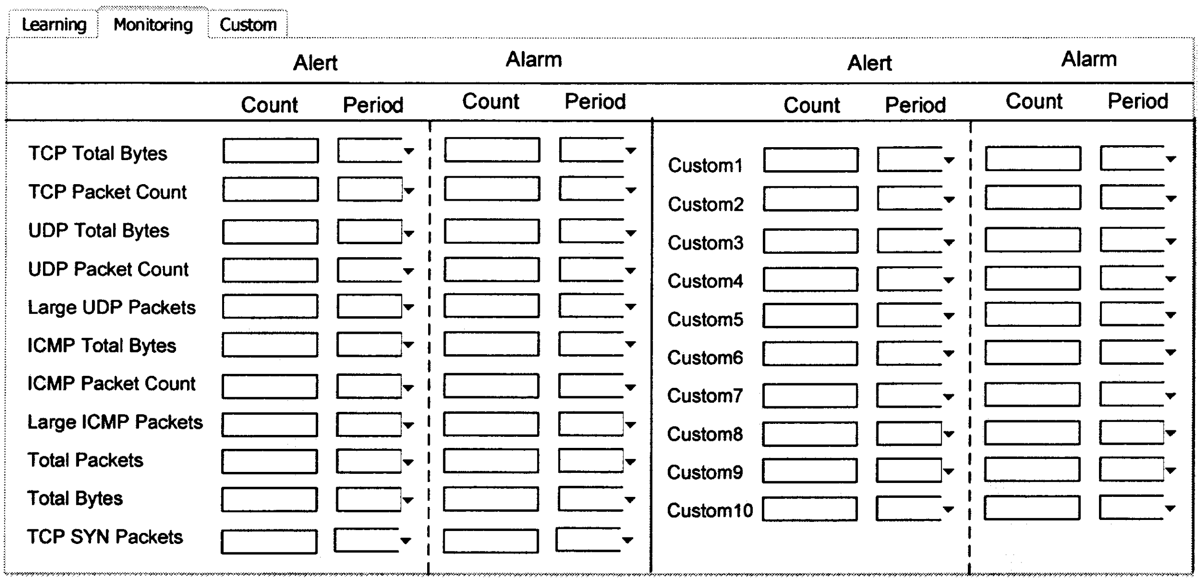


Figure 3.22 Proposed dialog box2.

3.3 Conclusion

Ongoing development of Conex intrusion detection mechanisms led to consistently higher results in live testing. Conex IDS algorithms have been shown to consistently reveal false positive, false positive, and miscalculation results of under one percent.

With security products regularly combining a number of different IDS technologies, a series of product enhancements were proposed. Improved communication between IDS devices will create a more complete security solution for customers of various sizes. The combination of a scalable communication mechanism and standards based messages will greatly enhance the potential of the Conex intrusion detection product.

Improvements in the product's ability to be customized by a security administrator will help to minimize the disadvantages related to training and tuning on a site specific basis. These optimization additions will help to create a more usable and scalable product better positioned to compete in a demanding market for security protection products.

REFERENCES

1. J. P. Anderson, "Computer security threat monitoring and surveillance," tech. rep., James P. Anderson Co., Fort Washington, PA, April 1980.

2. J. P. Anderson. "Computer Security Technology Planning Study Volume 2", October 1972, Available at HTTP: http://seclab.cs.ucdavis.edu/projects/history/papers/ande80.pdf

3. Carnegie Mellon Software Engineering Institute CERT Coordination Center. "CERT/CC Statistics 1988-2003." [2005Apr 16]. Available at HTTP: http://www.cert.org/stats/

4. The SANS^{TM} Institute, "SANS Intrusion Detection FAQ", [2005 April 18] Available at HTTP: http://www.sans.org/resources/idfaq/what_is_id.php

5. Common Intrusion Detection Framework, [2005 January 10] Available at HTTP: http://www.isi.edu/gost/cidf/drafts/architecture.txt

6. Tipping Point, [2004 November 8] Available at HTTP: http://www.tippingpoint.com

7. Z. Zhang, "Statistical Anomaly Denial of Service and Reconnaissance Intrusion Detection." May 2004

8. Internet Security Systems, [2004 November 8] Available at HTTP: http://www.iss.net/products_services/.

9. Symantec NetProwler and Intruder Alert, [2005 October 10] Available at HTTP: http://enterprisesecurity.symantec.com/products/products.cfm?ProductID=%50, 2002

10. Snort, [2004 November 8] Available at HTTP: http://www.snort.org.

11. Top Layer Intrusion Prevention System Products, [2005 October 10] Available at HTTP: http://www.toplayer.com/content/products/intrusion_detection/attack_mitigator.jsp

12. M. Castro, P. Druschel, A-M. Kermarrec and A. Rowstron, "SCRIBE: A large-scale and decentralised application-level multicast infrastructure", IEEE Journal on Selected Areas in Communication (JSAC), Vol. 20, No, 8, October 2002.

13. Juniper Networks ISG Series, [2005 October 10] Available at HTTP: http://www.juniper.net/products/integrated/ns_isg.html

14. C Krügel, T. Toth: "Using Decision Trees to Improve Signature-Based Intrusion Detection". RAID 2003: 173-191

15. K. Ilgun, R. Kemmerer, and P. Porras. State Transition Analysis: A RuleBased Intrusion Detection System. IEEE Transactions on Software Engineering, 21(3), Mar. 1995, [2005 October 10] Available at HTTP: http://citeseer.ist.psu.edu/ilgun95state.html

16. Bro Intrusion Detection System, [2005 October 10] Available at HTTP: http://www.bro-ids.org/Overview.html

17. A. K. Ghosh, C. Michael, M. Schatz: "A real-time intrusion detection system". RAID 2000: 93-109

18. A. Gupta,, and R. Sekar, "An approach for detecting self-propagating email using anomaly detection". RAID 2003: 55-72

19. A. Gupta,, and R. Sekar, "An approach for detecting self-propagating email using anomaly detection". RAID 2003: 55-72

20. G. Vigna, S. Eckmann, and R. Kemmerer. The STAT Tool Suite. In Proceedings of DISCEX 2000, Hilton Head, South Carolina, January 2000. IEEE Computer Society Press, [2005 October 10] Available at HTTP: http://citeseer.ist.psu.edu/vigna00stat.html

21. Bro: A System for Detecting Network Intruders in Real-Time, V. Paxson, Computer Networks, Volume 31, Numbers 23-24 (December 1999), pages 2435-2463.

22. Z. Zhang, "Statistical Anomaly Denial Of Service And Reconnaissance Intrusion Detection." May 2004

23. Tipping Point. IPS Statistics, [2005 October 10] Available at HTTP: http://www.tippingpoint.com/technology_threatsuppress.html.

24. Cisco: Configuring SPAN and RSPAN, [2005 October 10] Available at HTTP: http://www.cisco.com/univercd/cc/td/doc/product/lan/cat3750/12119eal/3750scg/swspan.htm#12866

25. CERT® Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks, [2005 October 10] Available at HTTP: http://www.cert.org/advisories/CA-1996-21.html

26. Centracomm Communications: Managed Netscreen Denial of Service Protection, [2005 October 10] Available at HTTP: http://www.centracomm.net/netscreen/dos.aspx

27. CERT® Advisory CA-1996-01 UDP Port Denial-of-Service Attack, [2005 October 10] Available at HTTP: http://www.cert.org/advisories/CA-1996-01.html

28. Advanced Networking Management Lab (ANML) Distributed Denial of Service Attacks(DDoS) Resources, [2004 October 10] Available at HTTP: http://www.anml.iu.edu/ddos/types.html.

29. CERT® Advisory CA-1998-01 Smurf IP Denial-of-Service Attacks, [2005 October 10] Available at HTTP: http://www.cert.org/advisories/CA-1998-01.html.

30. C. A. Huegen, "The Latest In Denial Of Service Attacks: "Smurfing" Description And Information To Minimize Effects." [2004 October 10] Available at HTTP: http://www.pentics.net/denial-of-service/white-papers/smurf.cgi. February 2000.

31. Cisco: Detecting DoS Attacks, [2004 October 10] Available at HTTP: http://www.ciscopress.com/articles/article.asp?p=345618

32. Denial of Service Attacks. Smack/Bloop, [2004 November 8] Available at HTTP: http://www.sabronet.com/dos/smackbloop.html

33. CERT® Advisory CA-1997-28 IP Denial-of-Service Attacks, [2004 November 8] Available at HTTP: http://www.cert.org/advisories/CA-1997-28.html.

34. Jamais vu un zine pareil!, [2004 November 8] Available at HTTP: http://membres.lycos.fr/hackworldclan2/lamah1.htm.

35. Auckland Department of Computer Science, [2004 November 8] Available at HTTP: http://www.tcs.auckland.ac.nz/~btech/btech2002/xxue002/resource/mid_report.do c

36. Insecure.org. Nmap Free Security Scanner, [2004 November 8] Available at HTTP: http://insecure.org

37. Hping security tool, [2004 November 8] Available at HTTP: http://www.hping.org.

38. H. Debar, D. Curry, B. Feinstein, The Intrusion Detection Message Exchange Format, [2004 November 8] Available at HTTP: http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-10.txt

39. McAfee IntruShield Network IPS Appliances, [2005 October 10] Available at HTTP: http://www.mcafee.com/us/products/mcafee/network_ips/intrushield_appliances.h tm

40. Fortigate-4000 Antivirus Firewall, [2005 October 10] Available at HTTP: http://www.firewalldepot.com/index.asp.

41. A. Rowstron and P. Druschel. "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. IFIP/ACM Middleware 2001*, Heidelberg, Germany, Nov. 2001.

42. M. Castro, P. Druschel, A-M. Kermarrec and A. Rowstron, "Scalable Application-level Anycast for Highly Dynamic Groups", NGC 2003, Munich, Germany, September 2003.

43. “Pastry - A scalable, decentralized, self-organizing and fault-tolerant substrate for peer-to-peer applications”, [2005 February 10], Available at HTTP: http://research.microsoft.com/~antr/Pastry/default.htm