DESIGN AND IMPLEMENTATION OF FIREWALL AND PACKET ANALYSIS

SUBMITTED IN THE PARTIAL FULFILLMENT FOR THE DEGREE OF

MASTER OF ENGINEERING

(ELECTRONICS & COMMUNICATION)

(2004-2006)

SUBMITED BY

SAMEER SHARMA (20/ E&C/ 2004)

(UNIVESITY ROLL NO: 8734)

UNDER THE GUIDENCE OF

SH. A.K.SINGH

ELECTRONICS & COMMUNICATION ENGINEERING

***DEPATMENT OF* ELECTRONICS & COMMUNICATION ENGINEERING**

DELHI COLLEGE OF ENGINEERING

UNIVERSITY OF DELHI

BAWANA ROAD, NEW DELHI

CERTIFICATE

THIS IS TO CERTIFY THAT THIS DISSERTATION TITLED "**A DESIGN AND IMPLEMENTATION OF FIREWALL AND PACKET ANALYSIS**" BEING SUBMITTED BY **SAMEER SHARMA (20/E&C/2004)** OF DELHI COLLEGE OF ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF ENGINEERING IN ELECTRONICS & COMMUNICATION IS A BONAFIDE WORK CARRIED OUT UNDER OUR GUIDANCE AND SUPERVISION.

SH.A.K.SINGH.

ELECTRONICS & COMMUNICATION
ENGINEERING DEPTT.
DELHI COLLAGE OF ENGINEERING

ACKNOWLEDGEMENTS

I AM HIGHLY INDEBTED AND EXPRESS MY DEEP SENSE OF GRATITUDE TO MY WORTHY AND REVEREND GUIDE **SH.A.K.SINGH,** , DEPTT. OF ELECTRONICS & COMMUNICATION ENGINEERING , DELHI COLLEGE OF ENGINEERING, NEW DELHI FOR THIS VALUABLE GUIDANCE AND HELP EXTENDED BY THIS WORTHINESS, WHICH HAVE ENABLED ME TO CARRIED OUT THE WORK SUCCESSFULLY. HIS COOPERATION CAME FORTH TO HELP ME OUT OF MY DIFFICULTIES BEFORE IT WAS EVEN CALLED FOR BY ME. HIS PRECIOUS SUGGESTION AND DEVOTION HAVE ENCOURAGED ME ALL THROUGH TO THE ADVANTAGE OF MAKING MY CARRIER BRIGHT.

I SHALL BE FAILING IN MY DUTY IF I DO NOT APPRECIATE AND APPLAUDED THE SERVICES OF AND THE TIME DEVOTED BY HONOR'ABLE **DR. A.BHATTACHARYA**, HEAD OF THE DEPTT. OF ELECTRICAL ENGINEERING. WITHOUT THE BLESSINGS, GOOD WISHES AND KIND HELP, THIS PROJECT WOULD NOT HAVE BEEN ACCOMPLISHED.

MY FAMILY HAD A BIG ROLL IN ENSURING THAT I SUCCESSFULLY COMPLETE THE PROJECT. THEIR CONSTANT ENCOURAGEMENT AND UNWAVERING SUPPORT HAVE BEEN A GREAT SOURCE OF STRENGTH FOR ME. I AM GRATEFUL TO MY FRIENDS FOR THEIR VALUABLE HELP AND GOOD WISHES WHICH WENT A LONG WAY IN FULFILLING THIS TASK.

SAMEER SHARMA

(ER.NO.20/E&C/2004, UNIV. R.NO. 8734)

CONTENTS

List of Abbreviation

1. IP:	Internet Protocol
2. MAC:	Media Access Control
3. NIC:	Network Interface Card
4. RFC:	Request for Comments
5. ARP:	Address Resolution Protocol
6. RARP:	Reverse Address Resolution Protocol
7. DLC:	Data Link Control
8. LLC:	Logical Link Control
9. TCP:	Transmission Control Protocol
10. UDP:	User Datagram Protocol
11. NAT:	Network Address Translation
12. VC:	Virtual Channel
13. VP:	Virtual Path
14. VPN:	Virtual Path Network
15. RSVP:	Resource Reservation Protocol

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION TO FIREWALL

NOT TOO LONG AGO, THE “FIREWALL” AS WE KNOW IT TODAY BELONGED TO AN EXCLUSIVE GROUP OF HIGHLY TRAINED NETWORK ENGINEERS AND PROGRAMMERS AS A TOOL SET FOR DEFENDING AN ORGANIZATION’S NETWORKS.

BACK THEN, A PERSON REALLY NEEDED TO KNOW WHAT HE WAS DOING TO SET UP A FIREWALL. LESS THAN A HANDFUL OF TODAY’S COMMERCIAL PRODUCTS WERE AVAILABLE, AND MOST NETWORK ENGINEERS AND PROGRAMMERS RESORTED TO BUILDING THEIR OWN FIREWALLS. THIS WAS BECAUSE, AT THE TIME, ONLY A FEW ORGANIZATIONS ON THE INTERNET HAD TO REALLY WORRY ABOUT PROTECTING THEMSELVES.

WELL, THAT WAS THEN. TODAY, MORE THAN 100 FIREWALL PROVIDERS ADVERTISE THEIR WARE. UNFORTUNATELY, MANY OF THESE SO-CALLED “FIREWALLS” AREN’T REALLY DOING AS MUCH AS YOU’D THINK.

TO REALLY UNDERSTAND HOW TO PROTECT NETWORKS, WE NEED TO FIRST UNDERSTAND WHAT A FIREWALL IS MADE UP OF. WITH THIS KNOWLEDGE, WE CAN INTELLIGENTLY DECIDE IF WE ARE BUILDING THE FIREWALL THAT SERVES OUR NEEDS.

NOTICE THAT I STILL REFER TO IT AS “BUILDING” A FIREWALL. MOST OF US DO NOT BUILD OUR OWN FIREWALLS ANYMORE. PERHAPS A FEW STILL DO, BUT THAT’S NO LONGER NECESSARY WITH TODAY’S TECHNOLOGY. WE MUST, HOWEVER, STILL CHOOSE THE RIGHT COMBINATION OF PRODUCTS THAT WILL SUCCESSFULLY PROTECT OUR NETWORKS.

NOT ALL FIREWALLS ARE CREATED EQUAL. AND NOT ALL FIREWALLS ARE DOING THE JOB YOU MAY THINK THEY’RE CREATED TO DO.

A FIREWALL SHOULD REALLY BE DOING MORE THAN SIMPLY FILTERING AND BLOCKING PARTICULAR NETWORK TRAFFIC. A GOOD FIREWALL SHOULD, AT THE MINIMUM, PROVIDE ADEQUATE SECURITY FOR ITS ORGANIZATION. HOWEVER, MOST FIREWALL MANUFACTURERS SEEM TO FORGET THAT GOOD SECURITY INCLUDES: RELIABILITY, PERFORMANCE AND MANAGEMENT.

BY MANAGEMENT, I MEAN PROVIDING INTELLIGENT INFORMATION ABOUT THE NETWORK AND FIREWALLS. MOST FIREWALLS TODAY ARE BEING SET UP AND FORGOTTEN. THEY’RE ABANDONED WITHOUT ROUTINE MAINTENANCES, PENETRATION TESTS AND AUDITING.

1.1 Definition of Firewall

A FIREWALL IS A SYSTEM DESIGNED TO PREVENT UNAUTHORIZED ACCESS TO OR FROM A PRIVATE NETWORK. FIREWALLS ARE NOW WIDELY USED BECAUSE OF THE VAST AMOUNT OF BROADBAND CONNECTIONS PRESENT. THEY PROVIDE A FIRST LINE OF DEFENSE FOR YOUR COMPUTER OR NETWORK. IF IT SUCCEEDS IN KEEPING THE BAD GUYS OUT, WHILE STILL LETTING YOU HAPPILY USE YOUR NETWORK, IT'S A GOOD FIREWALL[1]. EVERY CORPORATE NETWORK HAS AT LEAST ONE FIREWALL IN USE. FIREWALLS COME IN ALL SHAPES AND SIZES. MOST COMPUTERS ARE SHIPPED FROM THE FACTORY WITH SOME TYPE OF FIREWALL SOFTWARE OR MAY USE THE DEFAULT FIREWALL BUILT INTO XP

2. Types of Firewalls

THERE ARE TWO MAIN TYPES OF FIREWALLS: HARDWARE AND SOFTWARE. HIGH LEVEL HARDWARE FIREWALLS ARE VERY EXPENSIVE AND ARE NOT PRACTICAL FOR THE HOME USER. HOWEVER, LOW-END ROUTERS THAT PERFORM NAT ACT AS A HARDWARE FIREWALL. CHEAPER BROADBAND ROUTERS SUCH AS BELKIN, D-LINK, ETC PROVIDE THIS FUNCTIONALITY. IN A CORPORATE ENVIRONMENT, VERY EXPENSIVE DEVICES SUCH AS THE CISCO PIX, SYMANTEC FIREWALL, AND SONICWALL ARE COMMONLY USED HARDWARE SOLUTIONS. HARDWARE FIREWALLS ARE WITH A LARGE AMOUNT OF CLIENTS.

SOFTWARE FIREWALLS ARE PRACTICAL FOR HOME USERS BECAUSE THEY ARE NOTHING MORE THAN A PROGRAM THAT RUNS WITH YOUR OPERATING SYSTEM. THESE PROGRAMS ARE USUALLY INEXPENSIVE, FREE, OR COME BUILT INTO THE OPERATING SYSTEM. THEY ARE MANY DIFFERENT TYPES OF FIREWALLS AVAILABLE WITH MANY DIFFERENT OPTIONS.

2.1 Hardware vs. Software

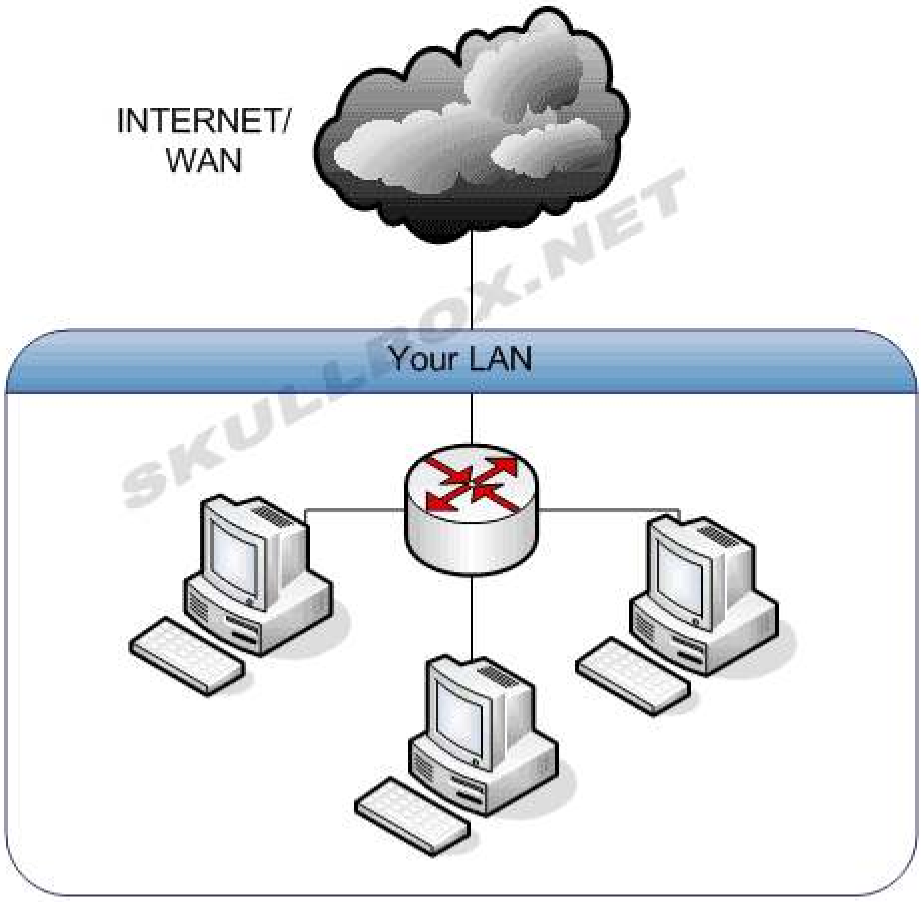
GENERALLY SPEAKING, HARDWARE FIREWALLS PERFORM BETTER THAN SOFTWARE FIREWALLS FOR SEVERAL REASONS. FIRST, A HARDWARE FIREWALL IS "DEDICATED" TO INSPECTING TRAFFIC. UNLIKE A SOFTWARE FIREWALL, IT DOES NOT COMPETE FOR CPU TIME OR RAM. THE MAIN DOWNSIDE TO A HARDWARE SOLUTION IS COST AND CONFIGURATION. HIGH-END DEVICES LIKE CISCO'S PIX FIREWALL CAN BE TRICKY TO CONFIGURE. SOFTWARE FIREWALLS ARE EASY TO INSTALL AND GENERALLY EASY TO CONFIGURE. IF YOU ARE ON A NETWORK WITH OTHER CLIENTS THAT YOU DO NOT KNOW, SUCH AS A DORM OR APARTMENT WITH SHARED INTERNET ACCESS, A SOFTWARE FIREWALL IS A MUST! REMEMBER THAT IF YOUR ROUTER PERFORMS NAT, IT ONLY PROTECTS YOU FROM INTERNET TRAFFIC. YOU ARE STILL VULNERABLE TO ATTACKS FROM WITHIN YOUR LAN. HACKERS OFTEN TARGET ROUTERS TO GAIN ACCESS TO OTHER DEVICES OR MACHINES ON A NETWORK.

LET US SAY SUSAN HAS A DSL CONNECTION AT HOME. FROM THERE SHE DOES HER BANKING, STOCK TRADING, AND OTHER PRIVATE COMMUNICATION. A FIREWALL IS IMPORTANT BECAUSE IT WOULD BLOCK CONNECTION ATTEMPTS BY A HACKER. IF A HACKER HAS SUSAN'S IP ADDRESS, HE CAN ESTABLISH A REMOTE CONNECTION. IF A SUCCESSFUL CONNECTION IS MADE, IT IS POSSIBLE FOR THAT HACKER TO INTERCEPT PASSWORDS OR OTHER DATA THAT ENDANGERS SUSAN'S ONLINE IDENTITY. ARE YOU WONDERING IF A FIREWALL IS FOR YOU? I'D

SAY "BETTER SAFE THAN SORRY." YOU WOULDN'T WANT TO OWN A STORE IN A DANGEROUS NEIGHBORHOOD WITHOUT A BURGLAR ALARM. BELOW ARE DIAGRAMS OF HOW A FIREWALLS CAN BE DEPLOYED IN A NETWORK:

Simple NAT Firewall

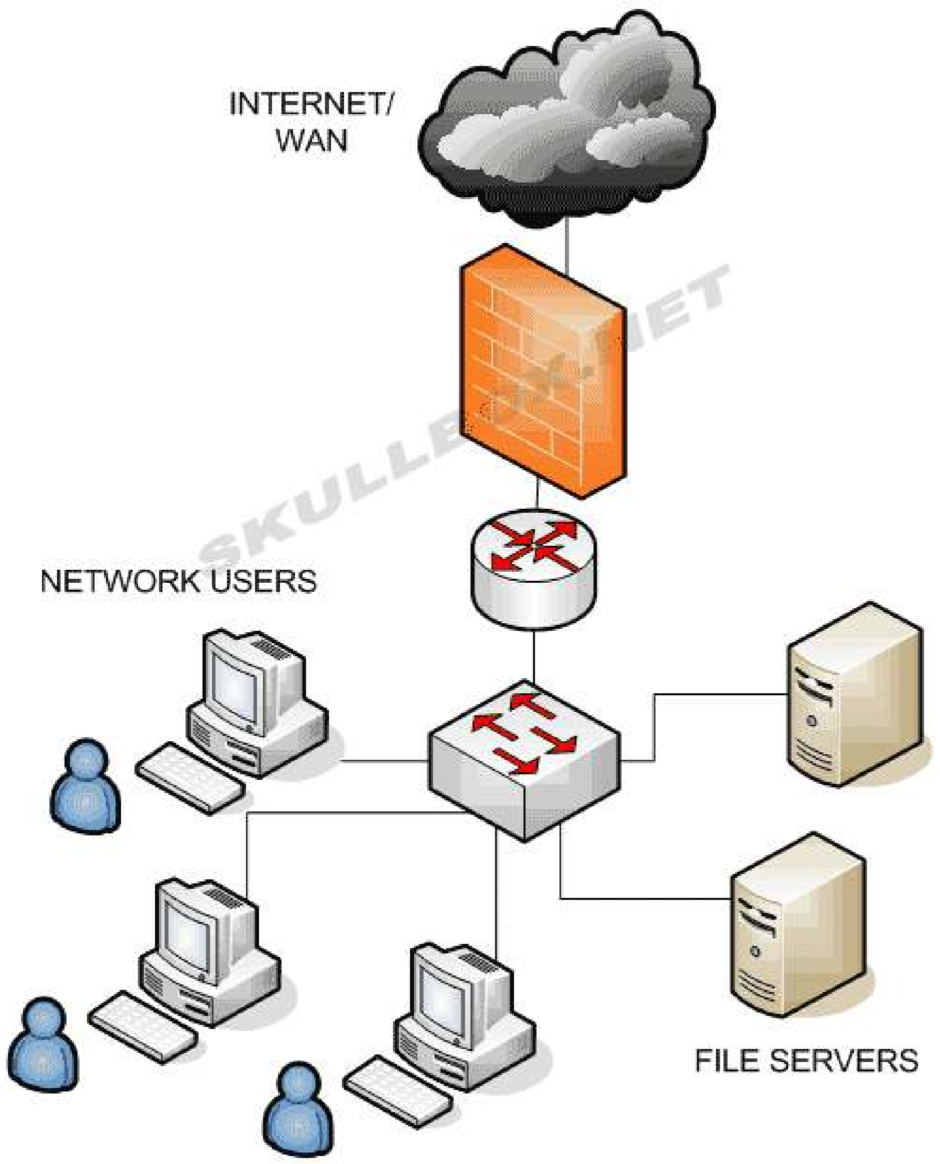
FIGURE 1.



THE DIAGRAM ABOVE ILLUSTRATES THE FIREWALL PROTECTION PROVIDED BY NAT. WHILE 3 MACHINES ARE ATTACHED TO THE ROUTER, THE INTERNET/WAN LINK THINKS ONLY 1 DEVICE IS PRESENT. THIS PROTECTS ALL 3 MACHINES BY LIMITING ACCESS TO IP ADDRESSES AND PORTS FROM THE INTERNET/WAN CONNECTION. FOR MORE DETAILS ON THIS SEE THE ARTICLE ON NAT.

2.3 Dedicated Firewall Device

Figure 2.



THE DIAGRAM ABOVE ILLUSTRATES HOW A DEDICATED FIREWALL DEVICE IS USED. NOTICE THAT THE FIREWALL PROTECTS THE ROUTER, SERVERS, AND NETWORK USERS. THIS IS A COMMON APPROACH USED WHEN ARE LARGE NUMBER OF USERS NEED TO ACCESS THE INTERNET. NAT IS NOT MEANT FOR LARGE NETWORKS. USING A DEDICATED FIREWALL DEVICE IN A HIGH-TRAFFIC ENVIRONMENT DOES NOT NEGATIVELY IMPACT NETWORK PERFORMANCE LIKE A SOFTWARE FIREWALL OR LOW-END NAT DEVICE.

3. Description of a Personal Firewall

3.1 Description of a Firewall

A FIREWALL IS DESIGNED TO HELP PROTECT YOUR COMPUTER FROM ATTACK BY MALICIOUS USERS OR BY MALICIOUS SOFTWARE SUCH AS VIRUSES THAT USE UNSOLICITED INCOMING NETWORK TRAFFIC TO ATTACK YOUR COMPUTER. BEFORE YOU DISABLE YOUR FIREWALL, YOU MUST DISCONNECT YOUR COMPUTER FROM ALL NETWORKS, INCLUDING THE INTERNET.

A FIREWALL IS A SYSTEM THAT IS DESIGNED TO PREVENT UNAUTHORIZED ACCESS TO OR FROM A PRIVATE NETWORK. YOU CAN IMPLEMENT FIREWALLS IN HARDWARE, SOFTWARE, OR BOTH. FIREWALLS ARE FREQUENTLY USED TO PREVENT UNAUTHORIZED INTERNET USERS FROM ACCESSING PRIVATE NETWORKS THAT ARE CONNECTED TO THE INTERNET

3.2 Different Types of Firewalls

DIFFERENT FIREWALLS USE DIFFERENT TECHNIQUES. MOST FIREWALLS USE TWO OR MORE OF THE FOLLOWING TECHNIQUES:

PACKET FILTERS: A PACKET FILTER LOOKS AT EACH PACKET THAT ENTERS OR LEAVES THE NETWORK AND ACCEPTS OR REJECTS THE PACKET BASED ON USER-DEFINED RULES. PACKET FILTERING IS FAIRLY EFFECTIVE AND TRANSPARENT, BUT IT IS DIFFICULT TO CONFIGURE. IN ADDITION, IT IS SUSCEPTIBLE TO IP SPOOFING.

APPLICATION GATEWAY: AN APPLICATION GATEWAY APPLIES SECURITY MECHANISMS TO SPECIFIC PROGRAMS, SUCH AS FTP AND TELNET. THIS TECHNIQUE IS VERY EFFECTIVE, BUT CAN CAUSE PERFORMANCE DEGRADATION.

CIRCUIT-LAYER GATEWAY: THIS TECHNIQUE APPLIES SECURITY MECHANISMS WHEN A TRANSMISSION CONTROL PROTOCOL (TCP) OR USER DATAGRAM PROTOCOL (UDP) CONNECTION IS ESTABLISHED. AFTER THE CONNECTION HAS BEEN ESTABLISHED, PACKETS CAN FLOW BETWEEN THE HOSTS WITHOUT FURTHER CHECKING.

PROXY SERVER: A PROXY SERVER INTERCEPTS ALL MESSAGES THAT ENTER AND LEAVE THE NETWORK. THE PROXY SERVER EFFECTIVELY HIDES THE TRUE NETWORK ADDRESSES.
APPLICATION PROXIES: APPLICATION PROXIES HAVE ACCESS TO THE WHOLE RANGE OF INFORMATION IN THE NETWORK STACK. THIS PERMITS THE PROXIES TO MAKE DECISIONS BASED ON BASIC AUTHORIZATION (THE SOURCE, THE DESTINATION, AND THE PROTOCOL), AND ALSO TO FILTER OFFENSIVE OR DISALLOWED COMMANDS IN THE DATA STREAM. APPLICATION PROXIES ARE "STATEFUL," MEANING THAT THEY KEEP THE "STATE" OF CONNECTIONS INHERENTLY. THE INTERNET CONNECTION FIREWALL FEATURE THAT IS INCLUDED IN WINDOWS XP IS A "STATEFUL" FIREWALL, AS WELL AS WINDOWS FIREWALL. WINDOWS FIREWALL IS INCLUDED IN WINDOWS XP SERVICE PACK 2 (SP2).

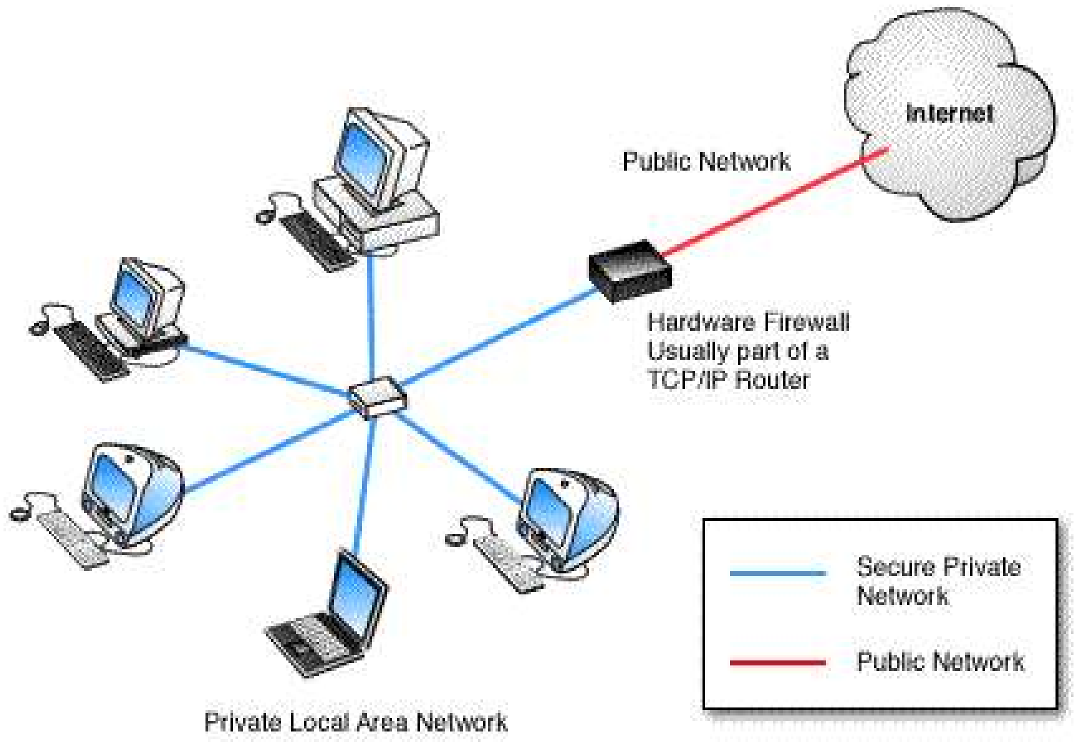
FOR ADDITIONAL INFORMATION ABOUT THE WINDOWS XP INTERNET CONNECTION FIREWALL FEATURE, CLICK THE ARTICLE NUMBER BELOW TO VIEW THE ARTICLE IN THE MICROSOFT KNOWLEDGE BASE:

FIREWALL

A firewall protects networked computers from intentional hostile intrusion that could compromise confidentiality or result in data corruption or denial of service. It may be a hardware device (fig 1 or a software programsee fig 2) running on a secure host computer. In either case, it must have at least two network interfaces, one for the network it is intended to protect, and one for the network it is exposed to. A firewall sits at the junction point or gateway between the two networks, usually a private network and a public network such as the Internet. The earliest firewalls were simply routers. The term firewall comes from the fact that by segmenting a network into different physical subnetworks, they limited the damage that could spread from one subnet to another just like firedoors or firewalls.

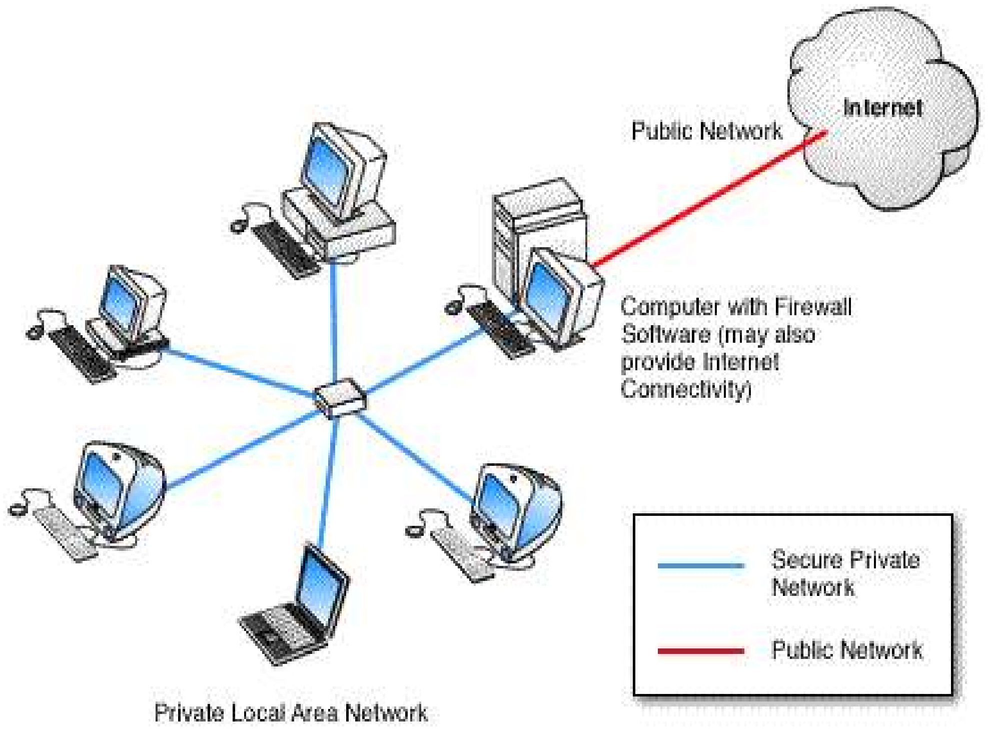
1 : Hardware Firewall

Figure 3.



2: Computer with Firewall Software.

Figure 4.


Internet
Public Network
Computer with Firewall Software (may also provide Internet Connectivity)
Secure Private Network
Public Network
Private Local Area Network

WHAT DOES A FIREWALL DO?

A firewall examines all traffic routed between the two networks to see if it meets certain criteria. If it does, it is routed between the networks, otherwise it is stopped. A firewall filters both inbound and outbound traffic. It can also manage public access to private networked resources such as host applications. It can be used to log all attempts to enter the private network and trigger alarms when hostile or unauthorized entry is attempted. Firewalls can filter packets based on their source and destination addresses and port numbers. This is known as address filtering. Firewalls can also filter specific types of network traffic. This is also known as protocol filtering because the decision to forward or reject traffic is dependant upon the protocol used, for example HTTP, ftp or telnet. Firewalls can also filter traffic by packet attribute or state.

WHAT CAN'T A FIREWALL DO?

A firewall cannot prevent individual users with modems from dialling into or out of the network, bypassing the firewall altogether. Employee misconduct or carelessness cannot be controlled by firewalls. Policies involving the use and misuse of passwords and user accounts must be strictly enforced. These are management issues that should be raised during the planning of any security policy but that cannot be solved with firewalls alone.

The arrest of the Phonemasters cracker ring brought these security issues to light. Although they were accused of breaking into information systems run by AT&T Corp., British Telecommunications Inc., GTE Corp., MCI WorldCom, Southwestern Bell, and Sprint Corp, the group did not use any high tech methods such as IP spoofing (see question 10). They used a combination of social engineering and dumpster diving. Social engineering involves skills not unlike those of a confidence trickster. People are tricked into revealing sensitive information. Dumpster diving or garbology, as the name suggests, is just plain old looking through company trash. Firewalls cannot be effective against either of these techniques.

WHO NEEDS A FIREWALL?

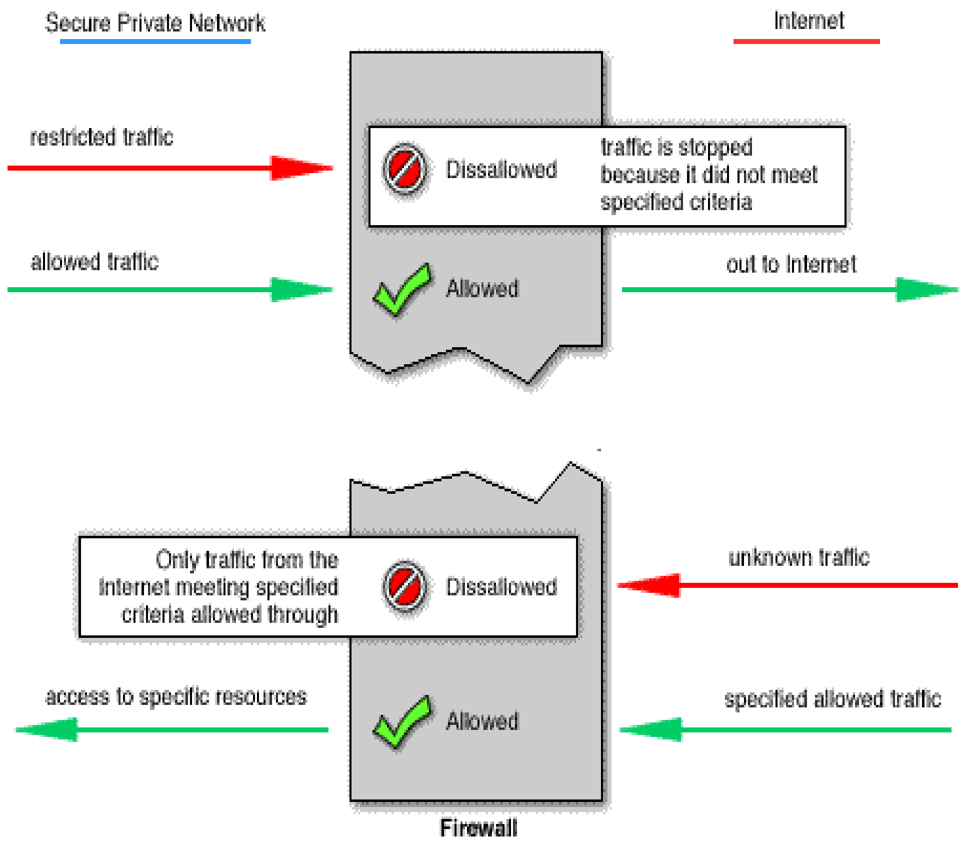
Anyone who is responsible for a private network that is connected to a public network needs firewall protection. Furthermore, anyone who connects so much as a single computer to the Internet via modem should have personal firewall software. Many dial-up Internet users believe that anonymity will protect them. They feel that no malicious intruder would be motivated to break into their computer. Dial up users who have been victims of malicious attacks and who have lost entire days of work, perhaps having to reinstall their operating system, know that this is not true. Irresponsible pranksters can use automated robots to scan random IP addresses and attack whenever the opportunity presents itself.

HOW DOES A FIREWALL WORK?

There are two access denial methodologies used by firewalls. A firewall may allow all traffic through unless it meets certain criteria, or it may deny all traffic unless it meets certain criteria (see figure 3). The type of criteria used to determine whether traffic should be allowed through varies from one type of firewall to another. Firewalls may be concerned with the type of traffic, or with source or destination addresses and ports. They may also use complex rule bases that analyse the application data to determine if the traffic should be allowed through. How a firewall determines what traffic to let through depends on which network layer it operates at. A discussion on network layers and architecture follows.

3: Basic Firewall Operation

Figure 5.



Secure Private Network
Internet
restricted traffic
Dissallowed
traffic is stopped because it did not meet specified criteria
allowed traffic
Allowed
out to Internet
Only traffic from the Internet meeting specified criteria allowed through
Dissallowed
unknown traffic
access to specific resources
Allowed
specified allowed traffic
Firewall

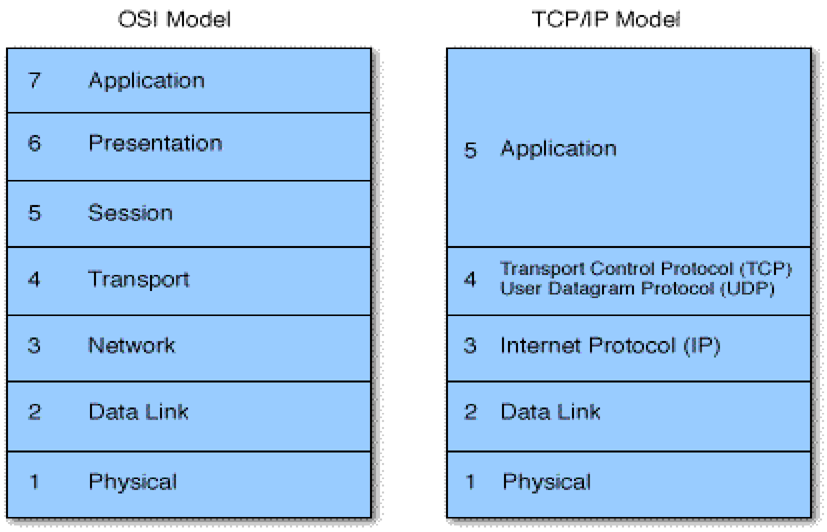
4. The OSI and TCP/IP Network Models

To understand how firewalls work it helps to understand how the different layers of a network interact. Network architecture is designed around a seven layer model. Each layer has its own set of responsibilities, and handles them in a well-defined manner. This enables networks to mix and match network protocols and physical supports. In a given network, a single protocol can travel over more than one physical support (layer one) because the physical layer has been dissociated from the protocol layers (layers three to seven). Similarly, a single physical cable can carry more than one protocol. The TCP/IP model is older than the OSI industry standard model which is why it does not comply in every respect. The first four layers are so closely analogous to OSI layers however that interoperability is a day to day reality.

Firewalls operate at different layers to use different criteria to restrict traffic. The lowest layer at which a firewall can work is layer three. In the OSI model this is the network layer. In TCP/IP it is the Internet Protocol layer. This layer is concerned with routing packets to their destination. At this layer a firewall can determine whether a packet is from a trusted source, but cannot be concerned with what it contains or what other packets it is associated with. Firewalls that operate at the transport layer know a little more about a packet, and are able to grant or deny access depending on more sophisticated criteria. At the application level, firewalls know a great deal about what is going on and can be very selective in granting access.

THE OSI AND TCP/IP MODELS

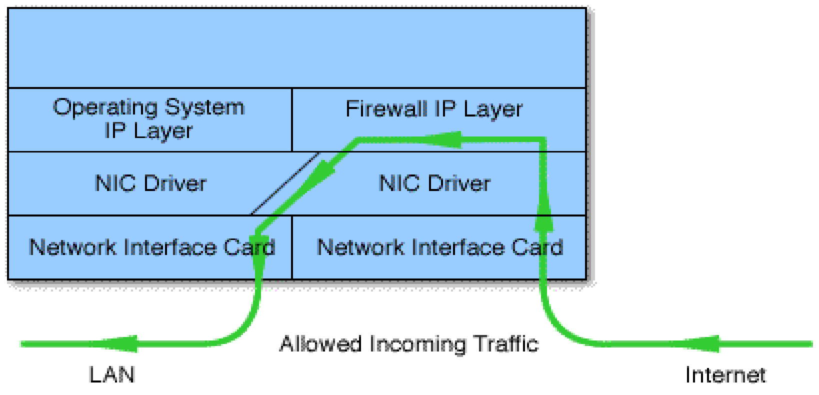
FIGURE 6.



It would appear then, that firewalls functioning at a higher level in the stack must be superior in every respect. This is not necessarily the case. The lower in the stack the packet is intercepted, the more secure the firewall. If the intruder cannot get past level three, it is impossible to gain control of the operating system.

Professional Firewalls Have Their Own IP Layer

Figure 7.



Professional firewall products catch each network packet before the operating system does, thus, there is no direct path from the Internet to the operating system's TCP/IP stack. It is therefore very difficult for an intruder to gain control of the firewall host computer then "open the doors" from the inside.

According To Byte Magazine*, traditional firewall technology is susceptible to misconfiguration on non-hardened OSes. More recently, however, "...firewalls have moved down the protocol stack so far that the OS doesn't have to do much more than act as a bootstrap loader, file system and GUI". The author goes on to state that newer firewall code bypasses the operating system's IP layer altogether, never permitting "potentially hostile traffic to make its way up the protocol stack to applications running on the system".

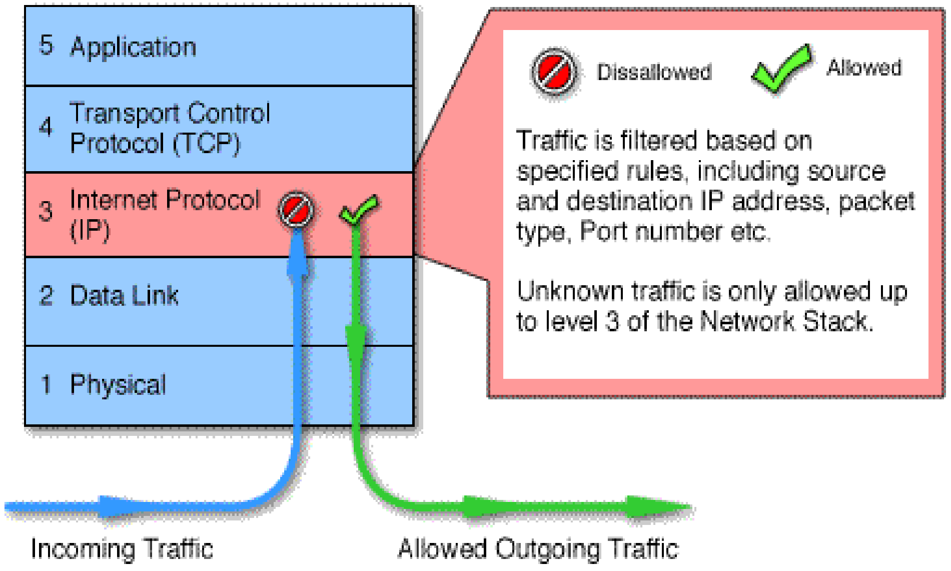
5. DIFFERENT TYPES OF FIREWALLS

Firewalls fall into four broad categories: packet filters, circuit level gateways, application level gateways and stateful multilayer inspection firewalls.

Packet filtering firewalls work at the network level of the OSI model, or the IP layer of TCP/IP. They are usually part of a router. A router is a device that receives packets from one network and forwards them to another network. In a packet filtering firewall each packet is compared to a set of criteria before it is forwarded. Depending on the packet and the criteria, the firewall can drop the packet, forward it or send a message to the originator. Rules can include source and destination IP address, source and destination port number and protocol used. The advantage of packet filtering firewalls is their low cost and low impact on network performance. Most routers support packet filtering. Even if other firewalls are used, implementing packet filtering at the router level affords an initial degree of security at a low network layer. This type of firewall only works at the network layer however and does not support sophisticated rule based models (see Figure 5). Network Address Translation (NAT) routers offer the advantages of packet filtering firewalls but can also hide the IP addresses of computers behind the firewall, and offer a level of circuit-based filtering.

5.1: Packet Filtering Firewall

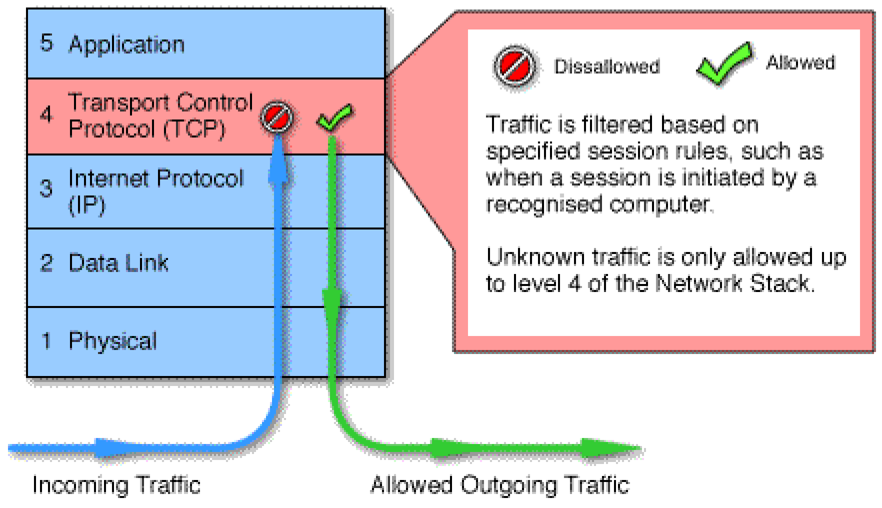
Figure 8.


5 Application
4 Transport Control Protocol (TCP)
3 Internet Protocol (IP)
2 Data Link
1 Physical
Dissallowed
Allowed
Traffic is filtered based on specified rules, including source and destination IP address, packet type, Port number etc.
Unknown traffic is only allowed up to level 3 of the Network Stack.
Incoming Traffic
Allowed Outgoing Traffic

Circuit level gateways work at the session layer of the OSI model, or the TCP layer of TCP/IP. They monitor TCP handshaking between packets to determine whether a requested session is legitimate. Information passed to remote computer through a circuit level gateway appears to have originated from the gateway. This is useful for hiding information about protected networks. Circuit level gateways are relatively inexpensive and have the advantage of hiding information about the private network they protect. On the other hand, they do not filter individual packets.

5.2 Circuit level Gateway

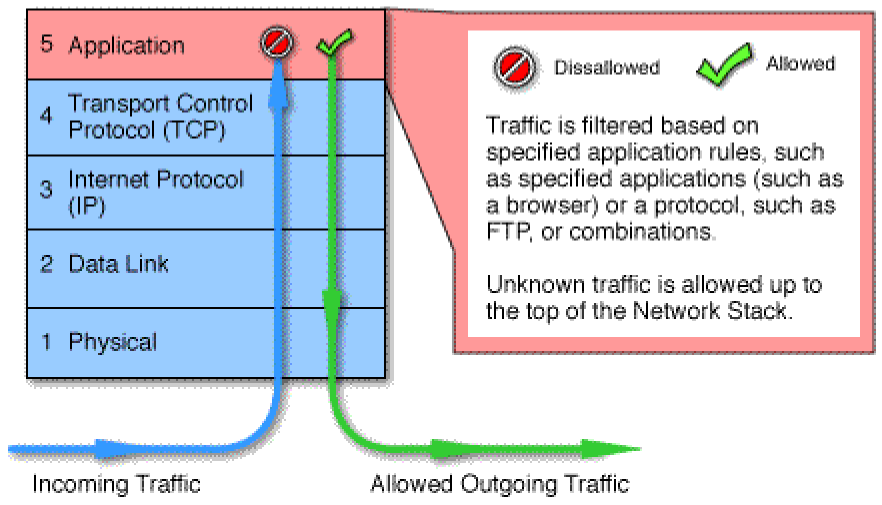
Figure 9.


5 Application
4 Transport Control Protocol (TCP)
3 Internet Protocol (IP)
2 Data Link
1 Physical
Dissallowed
Allowed
Traffic is filtered based on specified session rules, such as when a session is initiated by a recognised computer.
Unknown traffic is only allowed up to level 4 of the Network Stack.
Incoming Traffic
Allowed Outgoing Traffic

Application level gateways, also called proxies, are similar to circuit-level gateways except that they are application specific. They can filter packets at the application layer of the OSI model. Incoming or outgoing packets cannot access services for which there is no proxy. In plain terms, an application level gateway that is configured to be a web proxy will not allow any ftp, gopher, telnet or other traffic through. Because they examine packets at application layer, they can filter application specific commands such as http:post and get, etc. This cannot be accomplished with either packet filtering firewalls or circuit level neither of which know anything about the application level information. Application level gateways can also be used to log user activity and logins. They offer a high level of security, but have a significant impact on network performance. This is because of context switches that slow down network access dramatically. They are not transparent to end users and require manual configuration of each client computer.

5.3 Application level Gateway

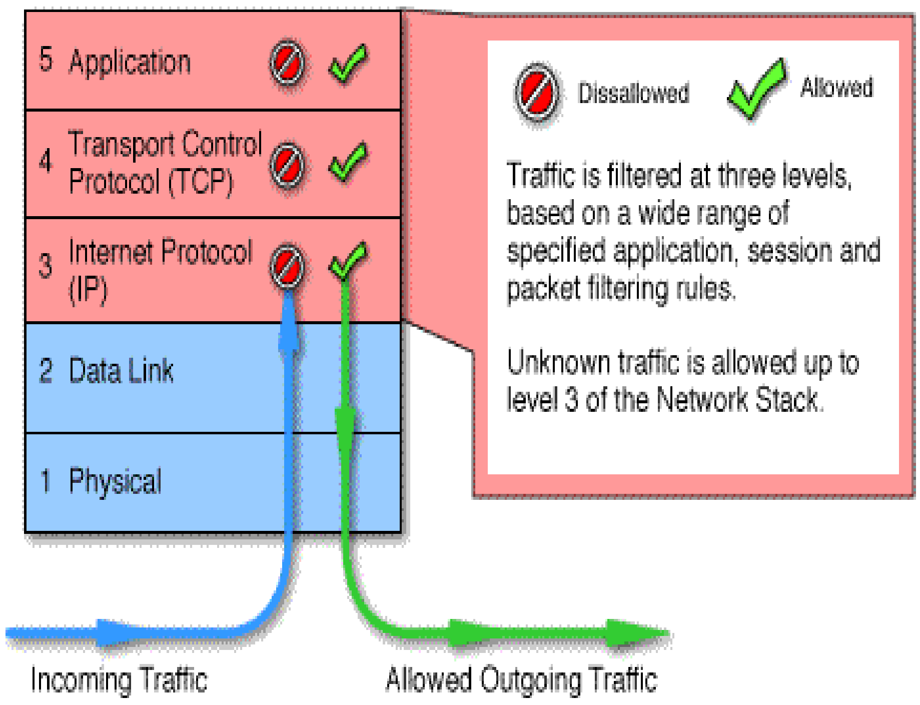
Figure 10.


5 Application
4 Transport Control Protocol (TCP)
3 Internet Protocol (IP)
2 Data Link
1 Physical
Dissallowed
Allowed
Traffic is filtered based on specified application rules, such as specified applications (such as a browser) or a protocol, such as FTP, or combinations.
Unknown traffic is allowed up to the top of the Network Stack.
Incoming Traffic
Allowed Outgoing Traffic

Stateful multilayer inspection firewalls combine the aspects of the other three types of firewalls. They filter packets at the network layer, determine whether session packets are legitimate and evaluate contents of packets at the application layer. They allow direct connection between client and host, alleviating the problem caused by the lack of transparency of application level gateways. They rely on algorithms to recognize and process application layer data instead of running application specific proxies. Stateful multilayer inspection firewalls offer a high level of security, good performance and transparency to end users. They are expensive however, and due to their complexity are potentially less secure than simpler types of firewalls if not administered by highly competent personnel.

5.4 Stateful Multilayer Inspection Firewall

Figure 11



5 Application
4 Transport Control Protocol (TCP)
3 Internet Protocol (IP)
2 Data Link
1 Physical
Dissallowed
Allowed
Traffic is filtered at three levels, based on a wide range of specified application, session and packet filtering rules.
Unknown traffic is allowed up to level 3 of the Network Stack.
Incoming Traffic
Allowed Outgoing Traffic

6.HOW DO I IMPLEMENT A FIREWALL?

We suggest you approach the task of implementing a firewall by going through the following steps:

a. DETERMINE THE ACCESS DENIAL METHODOLOGY TO USE.

It is recommended you begin with the methodology that denies all access by default. In other words, start with a gateway that routes no traffic and is effectively a brick wall with no doors in it.

b. DETERMINE INBOUND ACCESS POLICY.

If all of your Internet traffic originates on the LAN this may be quite simple. A straightforward NAT router will block all inbound traffic that is not in response to requests originating from within the LAN. As previously mentioned, the true IP addresses of hosts behind the firewall are never revealed to the outside world, making intrusion extremely difficult. Indeed, local host IP addresses in this type of configuration are usually non-public addresses, making it impossible to route traffic to them from the Internet. Packets coming in from the Internet in response to requests from local hosts are addressed to dynamically allocated port numbers on the public side of the NAT router. These change rapidly making it difficult or impossible for an intruder to make assumptions about which port numbers to use.

If your requirements involve secure access to LAN based services from Internet based hosts, then you will need to determine the criteria to be used in deciding when a packet originating from the Internet may be allowed into the LAN. The stricter the criteria, the more secure your network will be. Ideally you will know which public IP addresses on the Internet may originate inbound traffic. By limiting inbound traffic to packets originating from these hosts, you decrease the likelihood of hostile intrusion. You may also want to limit inbound traffic to certain protocol sets such as ftp or http. All of these techniques can be achieved with packet filtering on a NAT router. If you cannot know the IP addresses that may originate inbound traffic, and you cannot use protocol filtering then you will need more a more complex rule based model and this will involve a stateful multilayer inspection firewall.

c. DETERMINE OUTBOUND ACCESS POLICY.

If your users only need access to the web, a proxy server may give a high level of security with access granted selectively to appropriate users. As mentioned, however, this type of firewall requires manual configuration of each web browser on each machine. Outbound protocol filtering can also be transparently achieved with packet filtering and no sacrifice in security. If you are using a NAT router with no inbound mapping of traffic originating from the Internet, then you may allow LAN users to freely access all services on the Internet with no security compromise. Naturally, the risk of employees behaving irresponsibly with email or with external hosts is a management issue and must be dealt with as such.

D. DETERMINE IF DIAL-IN OR DIAL-OUT ACCESS IS REQUIRED.

Dial-in requires a secure remote access PPP server that should be placed outside the firewall. If dial-out access is required by certain users, individual dial-out computers must be made secure in such a way that hostile access to the LAN through the dial-out connection becomes impossible. The surest way to do this is to physically isolate the computer from the LAN. Alternatively, personal firewall software may be used to isolate the LAN network interface from the remote access interface.

DECIDE WHETHER TO BUY A COMPLETE FIREWALL PRODUCT, HAVE ONE IMPLEMENTED BY A SYSTEMS INTEGRATOR OR IMPLEMENT ONE YOURSELF.

Once the above questions have been answered, it may be decided whether to buy a complete firewall product or to configure one from multipurpose routing or proxy software. This decision will depend as much on the availability of in-house expertise as on the complexity of the need. A satisfactory firewall may be built with little expertise if the requirements are straightforward. However, complex requirements will not necessarily entail recourse to external resources if the system administrator has sufficient grasp of the elements. Indeed, as the complexity of the security model increases, so does the need for in-house expertise and autonomy.

7.Firewall related problems

Firewalls introduce problems of their own. Information security involves constraints, and users don't like this. It reminds them that Bad Things can and do happen. Firewalls restrict access to certain services. The vendors of information technology are constantly telling us "anything, anywhere, any time", and we believe them naively. Of course they forget to tell us we need to log in and out, to memorize our 27 different passwords, not to write them down on a sticky note on our computer screen and so on.

Firewalls can also constitute a traffic bottleneck. They concentrate security in one spot, aggravating the single point of failure phenomenon. The alternatives however are either no Internet access, or no security, neither of which are acceptable in most organizations.

8.BENEFITS OF A FIREWALL

Firewalls protect private local area networks from hostile intrusion from the Internet. Consequently, many LANs are now connected to the Internet where Internet connectivity would otherwise have been too great a risk.

Firewalls allow network administrators to offer access to specific types of Internet services to selected LAN users. This selectivity is an essential part of any information management program, and involves not only protecting private information assets, but also knowing who has access to what. Privileges can be granted according to job description and need rather than on an all-or-nothing basis.

9. What is a MAC Address.

A MAC ADDRESS IS A UNIQUE NUMBER ASSIGNED TO A NETWORK INTERFACE CARD (NIC), COMMONLY CALLED AN ETHERNET CARD. THIS "ADDRESS" IS CREATED BY THE MANUFACTURER (NOT BY WASHINGTON UNIVERSITY). A MAC ADDRESS IS A 12-DIGIT NUMBER. EACH DIGIT IS A NUMBER FROM 0-9 OR A LETTER FROM A-F. SOMETIMES THE DIGITS OF A MAC ADDRESS ARE SEPARATED BY COLONS OR DASHES. EXAMPLES OF POSSIBLE MAC ADDRESSES INCLUDE: 080007A92BFC, 09:00:07:A9:B2:EB, OR 09-10-4A-B9-E2-A4.

9.1 IMPORTANCE

DON'T CONFUSE THE MAC ADDRESS WITH APPLE OR MACINTOSH COMPUTERS, WHICH ARE COMMONLY REFERRED TO AS "MACS." THE NAME "MAC ADDRESS" DOES NOT REFER TO APPLE/MACINTOSH COMPUTERS, BUT INSTEAD ONLY TO THE PHYSICAL ADDRESS OF YOUR COMPUTER, REGARDLESS OF WHETHER IT IS A PC OR A MACINTOSH.

YOUR MAC ADDRESS MAY ALSO BE CONFUSED WITH AN INTERNET PROTOCOL (IP) ADDRESS OR AN E-MAIL ADDRESS. AN IP ADDRESS USES ONLY NUMBERS AND PERIODS: 128.252.93.1

YOUR MAC ADDRESS MAY ALSO LOOK LIKE A MODEM ADDRESS. HOWEVER, MODEM ADDRESS DESCRIPTION WILL BE TITLED "PPP" OR "MODEM."

9.2 Obtaining Your MAC Address

If you cannot find your MAC address in either the box or on the card, please follow the instructions below, depending on your operating system (Windows 95, Windows 98, Windows ME, Windows NT, Windows 2000 Professional, Win XP, Macintosh OS, Macintosh OS X, Solaris/SunOS, Linux, FreeBSD, or HP).

Linux

On Linux systems, the ethernet device is typically called eth0. In order to find the MAC address of the ethernet device, you must first become root, through the use of su. Then, type **ifconfig -a** and look up the relevant info. For example:

ifconfig -a eth0 Link encap:Ethernet HWaddr 00:60:08:C4:99:AA inet addr:131.225.84.67 Bcast:131.225.87.255 Mask:255.255.248.0 UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1 RX packets:15647904 errors:0 dropped:0 overruns:0 TX packets:69559 errors:0 dropped:0 overruns:0 Interrupt:10 Base address:0x300

The MAC address is the HWaddr listed on the first line. In the case of this machine, it is 00:60:08:C4:99:AA.

9.3 IP Masquerading or IP MASQ

This document describes how to enable the Linux IP Masquerade feature on a given Linux host. IP Masquerade, called "IPMASQ" or "MASQ" for short, is a form of Network Address Translation (NAT) which allows internally connected computers that do not have one or more registered Internet IP addresses to communicate to the Internet via the Linux server's Internet IP address. Since IPMASQ is a generic technology, you can connect the Linux server's internal and external to other computers through LAN technologies like Ethernet, TokenRing, and FDDI, as well as dialup connections line PPP or SLIP links. This document primarily uses Ethernet and PPP connections in examples because it is most commonly used with DSL / Cablemodems and dialup connections.

This document is intended for systems running stable Linux kernels like 2.4.x, 2.2.x, and 2.0.x preferably on an IBM-compatible PC. IP Masquerade does work on other Linux-supported platforms like Sparc, Alpha, PowerPC, etc. but this HOWTO doesn't cover them in as much detail. Beta kernels such as 2.5.x, 2.3.x, 2.1.x, and ANY kernels less than 2.0.x are NOT covered in this document. The primary reason for this is because many of the older kernels are considered broken. If you are using an older kernel version, it is highly advisable to upgrade to one of the stable Linux kernels before using IP Masquerading.

9.4 Foreword, Feedback & Credits

As a new user, I found it very confusing to setup IP masquerade on the Linux kernel, (back then, its was a 1.2.x kernel). Although there was a FAQ and a mailing list, there was no documentation dedicated to this. There was also some requests on the mailing list for a HOWTO manual. So, I decided to write this HOWTO as a starting point for new users and possibly create a building block for other knowledgeable users. If you, the reader, have any additional ideas, corrections, or questions about this document, please feel free to contact us.

This document was originally written by Ambrose Au back in August, 1996, based on the 1.x kernel IPMASQ FAQ written by Ken Eves and numerous helpful messages from the original IP Masquerade mailing list. In particular, a mailing list message from Matthew Driver inspired Ambrose to set up IP Masquerade and eventually write version 0.80 of this HOWTO. In April 1997, Ambrose created the Linux IP Masquerade Resource Web site at which has provided up-to-date information on Linux IP Masquerading ever since. In February 1999, David Ranch took over maintenance of the HOWTO. David then re-wrote the HOWTO and added a substantial number of sections to the document. Today, the HOWTO is still maintained by David where he constantly updates it and fixes any reported bugs, etc.

Please feel free to send any feedback or comments regarding this HOWTO to if you have any corrections or if any information/URLs/etc. is missing. Your invaluable feedback will certainly influence the future of this HOWTO!

This HOWTO is meant to be a fairly comprehensive guide to getting your Linux IP Masquerading system working in the shortest time possible. David only plays a technical writer on T.V. so you might find the information in this document not as general and/or objective as it could be. If you think a section could be clearer, etc.. please let David know. The latest version of the MASQ HOWTO can be found at Additional news, mirrors of the HOWTO, and information regarding IPMASQ can be found at the *IP Masquerade Resource* web page. If you have any technical questions on IP Masquerade, please join the *IP Masquerade Mailing List* instead of sending email to David or Ambrose. Most MASQ problems are -common- for ALL MASQ users and can be easily solved by users on the list. In addition to this, the response time of the IP MASQ email list will be much faster than a reply from either David or Ambrose

9.5 How does IP Masquerade Work

Based from the original IP Masquerade FAQ by Ken Eves: Here is a drawing of the most simplistic setup:

```
PPP/ETH/etc.       +------------+                     +-------------+
to ISP provider    |  Linux #1  |      PPP/ETH/etc.    | Anybox      |
                |           |                    |           |
 <---------- modem1|         |modem2 ----------- modem3|           |
                |           |                    |           |
  111.222.121.212 |         |        192.168.0.100 |           |
                +------------+                     +-------------+
```

In the above drawing, a Linux box with IP_MASQUERADING is installed as Linux #1 and is connected to the Internet via PPP, Ethernet, etc. It has an assigned public IP address of 111.222.121.212. It also has another network interface (e.g. modem2) connected to allow incoming network traffic be it from a PPP connection, Ethernet connection, etc.

The second system (which does not need to be Linux) connects into the Linux #1 box and starts its network traffic to the Internet. This second machine does NOT have a publicly assigned IP address from the Internet, so it uses an RFC1918 private address, say 192.168.0.100. (see below for more info)

With IP Masquerade and the routing configured properly, this second machine "Anybox" can interact with the Internet as if it was directly connected to the Internet with a few small exceptions [noted later].

Quoting Pauline Middelink (the founder of Linux's IPMASQ):

"Do not forget to mention that the "ANYBOX" machine should have the Linux #1 box configured as its default gateway (whether it be the default route or just a subnet is no matter). If the "ANYBOX" machine is connected via a PPP or SLIP connection, the Linux #1 machine should be configured to support proxy arp for all routed addresses. But, the setup and configuration of proxy arp is beyond the scope of this document. Please see the PPP-HOWTO for more details."

The following is an excerpt on how IPMASQ briefly works though this will be explained in more detail later. This short text is based from a previous post on comp.os.linux.networking which has been edited to match the names used in the above example:

```
o I tell machine ANYBOX that my PPP or Ethernet connected Linux box is its
  gateway.

o When a packet comes into the Linux box from ANYBOX, it will assign the
  packet to a new TCP/IP source port number and insert its own IP address
  inside the packet header, saving the originals.  The MASQ server will
  then send the modified packet over the PPP/ETH interface onto the
  Internet.

o When a packet returns from the Internet into the Linux box, Linux
  examines if the port number is one of those ports that was assigned
  above.  If so, the MASQ server will then take the original port and
  IP address, put them back in the returned packet header, and send
  the packet to ANYBOX.

o The host that sent the packet will never know the difference.
```

:A typical example is given in the diagram below:

```
              Ethernet
             192.168.0.x
 +----------+
 |          |
 | A-box    |::::::
 |          |.2   :
 +----------+     :
                  :     +----------+   PPP/ETH
 +----------+     :   .1 |  Linux   |     link
 |          |     ::::::| Masq-Gate|:::::::::::::::::::>> Internet
 | B-box    |::::::      |          |  111.222.121.212
 |          |.3   :      +----------+
 +----------+     :
                  :
 +----------+     :
 |          |     :
 | C-box    |::::::
 |          |.4
 +----------+


 |                      |          |                          >
 | <-Internal Network--> |          | <- External Network ----> >
 |   connected via an    |          |    Connected from the     >
 |   Ethernet hub or     |          |    Linux server to your   >
 |       switch          |          |    Internet connection    >
```

In this example, there are (4) computer systems that we are concerned about. There is also presumably something on the far right that your PPP/ETH connection to the Internet comes through (modem server, DSL DSLAM, Cablemodem router, etc.). Out on the Internet, there exists some remote host (very far off to the right of the page) that you are interested in communicating with). The Linux system named *Masq-Gate* is the IP Masquerading gateway for ALL internal networked machines. In this example, the machines *A-box*, *B-box*, and *C-box* would have to go through the Masq-Gate to reach the Internet. The internal network uses one of several RFC-1918 assigned private network addresses, where in this case, would be the Class-C network 192.168.0.0. If you aren't familiar with RFC1918, it is encouraged to read the first few chapters of the RFC but the jist of it is that the TCP/IP addresses 10.0.0.0/8, 172.16-31.0.0/12, and 192.168.0.0/16 are reserved. When we say "reserved", we mean that anyone can use these addresses as long as they aren't routed over the Internet. ISPs are even allowed to use this private addressing space as long as they keep these addresses within their own networks and NOT advertise them to other ISPs. Unfortunately, this isn't always the case but thats beyond the scope of this HOWTO.

Anyway, the Linux box in the diagram above has the TCP/IP address 192.168.0.1 while the other systems has the addresses:

- A-Box: 192.168.0.2
- B-Box: 192.168.0.3
- C-Box: 192.168.0.4

The three machines, A-box, B-box and C-box, can have any one of several operating systems, just as long as they can speak TCP/IP. Some such as *Windows 95*, *Macintosh MacTCP or OpenTransport* , or even another *Linux box* have the ability to connect to other machines on the Internet. When running the IP Masquerade, the masquerading system or MASQ-gate converts all of these internal connections so that they appear to originate from the masq-gate itself. MASQ then arranges so that the data coming back to a masqueraded connection is relayed to the proper originating system. Therefore, the systems on the internal network are only able to see a direct route to the internet and are unaware that their data is being masqueraded. This is called a "Transparent" connection.

- The differences between NAT, MASQ, and Proxy servers.
- How packet firewalls work

9.6 Requirements for IP Masquerade on Linux 2.4.x

The newest 2.4.x kernels are now using both a completely new TCP/IP network stack as well as a new NAT sub-system called NetFilter. Within this NetFilter suite of tools, we now have a tool called IPTABLES for the 2.4.x kernels much like there was IPCHAINS for the 2.2.x kernels and IPFWADM for the 2.0.x kernels. The new IPTABLES system is far more powerful (combines several functions into one place like true NAT functionality), offers better security (stateful inspection), and better performance with the new 2.4.x TCP/IP stack. But this new suite of tools can be a bit complicated in comparison to older generation kernels. Hopefully, if you follow along with this HOWTO carefully, setting up IPMASQ won't be too bad. If you find anything unclear, downright wrong, etc. please email David about it.

Unlike the migration to IPCHAINS from IPFWADM, the new NetFilter tool has kernel modules that can actually support older IPCHAINS and IPFWADM rulesets with minimal changes. So re-writing your old MASQ or firewall ruleset scripts is not longer required. **BUT..** with the 2.4.x kernels, you cannot use the old 2.2.x MASQ modules like ip_masq_ftp, ip_masq_irc, etc. **AND** IPCHAINS is incompatible with the new IPTABLES modules like ip_conntrack_ftp, etc. So, what does this mean? It basically means that if you want to use IPMASQ or PORTFW functionality under a 2.4.x kernel, you shouldn't use IPCHAINS rules but IPTABLES ones instead. Please also keep in mind that there might be several benefits in performing a full ruleset re-write to take advantage of the newer IPTABLES features like stateful tracking, etc. but that is dependant upon how much time you have to migrate your old rulesets..

Some new 2.4.x functionalities include the following:

```
Status   = Module name =      Description and notes
---------   -----------   ---------------------------------
 Ported     CuSeeme      Used for Video conferencing

NotPorted   DirectPlay    Used for online Microsoft-based games

 Ported       FTP       Used for file transfers
                       - NOTEs:  Built into the kernel and
                                 fully supports PORTFWed FTP

ReWritten     H.323       Used for Video conferencing

NotPorted      ICQ       Used for Instant messaging
                       * No longer required for modern ICQ clients

 Ported       Irc       Used for Online chat rooms

 Ported      Quake       Used for online Quake games

 Ported       PPTP       Allow for multiple clients to the same server

NotPorted   Real Audio    Used for Streaming video / audio
                       * No longer required for modern RealVideo clients

NotPorted    VDO Live     Used for Streaming audio?
```

Documentation on how to perform MASQ module porting is available at If you have the time and knowledge, your talent would highly be appreciated in porting these modules.

If you'd like to read up more on NetFilter and IPTables, please see: and more specifically

Linux 2.4.x IP Masquerade requirements include:

Any decent computer hardware. See for more details.

The 2.4.x kernel source is available from

NOTE: Most modern Linux distributions, that natively come with 2.4.x kernels are typically modular kernels and have all the IP Masquerade functionality already included. In such cases, there is no need to compile a new Linux kernel. If you are UPGRADING your kernel, you should be aware of other programs that might be required and/or need to be upgraded as well (mentioned later in this HOWTO).

The program "iptables" version 1.2.4 or newer (1.2.7a or newer is highly recommended) archive available from.

NOTE #1: All versions of IPTABLES less than 1.2.3 have a FTP module issue that can bypass any existing firewall rulesets. ALL IPTABLES users are highly recommended to upgrade to the newest version. The URL is above.

NOTE #2: All versions of IPTABLES less than 1.2.2 have a FTP "port" security vulnerability in the ip_conntrack_ftp module. All IPTABLES users are highly recommended to upgrade to the newest version. The URL is above.

This tool, much like the older IPCHAINS and IPFWADM tools enables the various Masquerding code, more advanced forms of NAT, packet filtering, etc. It also makes use of additional MASQ modules like the FTP and IRC modules. Additional information on version requirements for the newest IPTABLES howto, etc. is located at the Unreliable IPTABLES HOWTOs page.

Loadable kernel modules, preferably 2.1.121 or higher, are available from A properly configured and running TCP/IP network running on the Linux machine as covered in Linux NET HOWTO and the Network Administrator's Guide . Also check out the TrinityOS document which is also authored by David Ranch. TrinityOS is a very comprehensive guide for Linux networking. Some topics include IP MASQ, security, DNS, DHCP, Sendmail, PPP, Diald, NFS, IPSEC-based VPNs, and performance sections, to name a few. There are over Fifty sections in all!

2.7. Requirements for IP Masquerade on Linux 2.2.x

NOTE: Most modern Linux distributions, that natively come with 2.2.x kernels are typically modular kernels and have all the IP Masquerade functionality already included. In such cases, there is no need to compile a new Linux kernel. If you are UPGRADING your kernel, you should be aware of other programs that might be required and/or need to be upgraded as well (mentioned later in this HOWTO).

NOTE #1: --- UPDATE YOUR KERNEL --- Linux 2.2.x kernels less than version 2.2.20 contain several different security vulnerabilities (some were MASQ specific). Kernels less than 2.2.20 have a few local vulnerabilities. Kernel versions less than 2.2.16 have a TCP root exploit vulnerability and versions less than 2.2.11 have a IPCHAINS fragmentation bug. Because of these issues, users running a firewall with strong IPCHAINS rulesets are open to possible instrusion. Please upgrade your kernel to a fixed version.

NOTE #2: Some newer _ such as Redhat 5.2 might not be Linux 2.2.x ready (upgradable). Tools like DHCP, NetUtils, etc. will need to be upgraded. More details can be found later in the HOWTO.

Loadable kernel modules, preferably 2.1.121 or higher, are available from

A properly configured and running TCP/IP network running on the Linux machine as covered in Linux NET HOWTO and the Network Administrator's Guide . Also check out the TrinityOS document which is also authored by David Ranch. TrinityOS is a very comprehensive guide for Linux networking. Some topics include IP MASQ, security, DNS, DHCP, Sendmail, PPP, Diald, NFS, IPSEC-based VPNs, and performance sections, to name a few. There are over Fifty sections in all!

10. PACKET ANALYSIS AND FILTERING

Table 1.

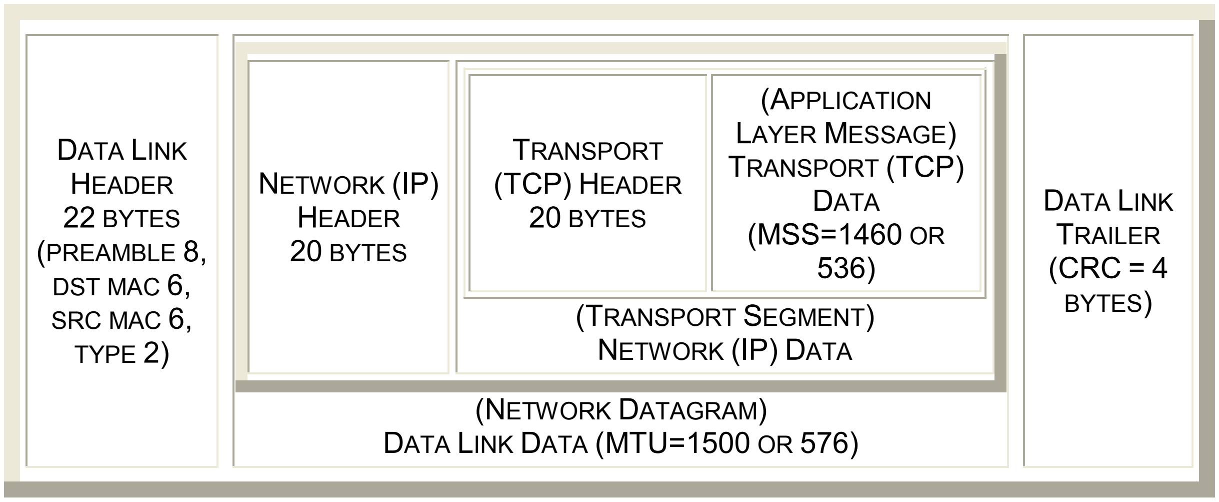
MAC header	**IP header**	**TCP header**	**Data :::**

Here our task is capture the live packet on the network and analyze the packet. Analysis means we have to read the whole contents of the packet. Packets compose of different fields like MAC header, IP address, TCP header and data. Here we have to read all fields.

Packet filtering means we have to filter the packet based on some critera, which may be ip address, Mac address and port number.

Suppose we have to monitor yahoo messenger packet than we have to monitor 5050 port. So we have to filter packet based on port 5050 (filtering based on port number) .
Than we are able to analyze the whole data.

For Capturing and Analyze the data we are using PCAP library. We are also using tcpdump and tcpslice library

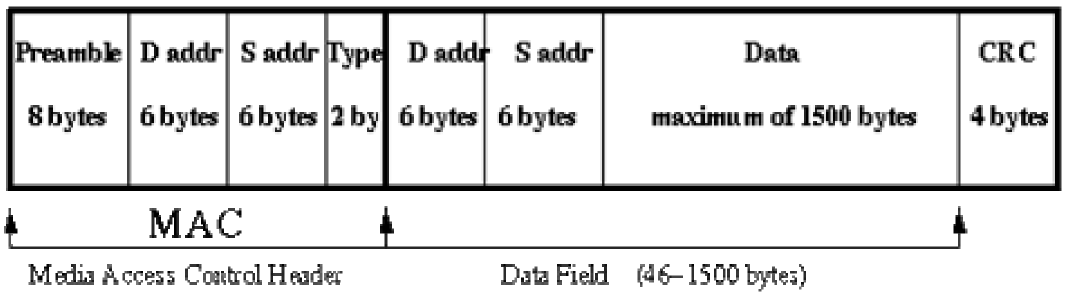

Data Link Header 22 bytes (preamble 8, dst MAC 6, src MAC 6, type 2)
Network (IP) Header 20 bytes
Transport (TCP) Header 20 bytes
(Application Layer Message) Transport (TCP) Data (MSS=1460 or 536)
(Transport Segment) Network (IP) Data
(Network Datagram) Data Link Data (MTU=1500 or 576)
Data Link Trailer (CRC = 4 bytes)

11. Application Data, TCP Header, IP header, and Ethernet Header in an Ethernet (Data Link Layer) Frame

ETHERNET FRAME: ---

Ethernet traffic is transported in units of a frame, where each frame has a definite beginning and end. The form of the frame is in the figure below.

TABLE 2.



- PREAMBLE FIELD USED FOR SYNCHRONIZATION, 64-BITS
- DESTINATION ADDRESS ETHERNET ADDRESS OF THE DESTINATION HOST, 48-BITS
- SOURCE ADDRESS ETHERNET ADDRESS OF THE SOURCE HOST, 48-BITS
- TYPE TYPE OF DATA ENCAPSULATED, E.G. IP, ARP, RARP, ETC, 16-BITS.
- DATA FIELD DATA AREA, 46-1500 BYTES, WHICH HAS
  DESTINATION ADDRESS INTERNET ADDRESS OF DESTINATION HOST
  SOURCE ADDRESS INTERNET ADDRESS OF SOURCE HOST
- CRC CYCLICAL REDUNDANCY CHECK, USED FOR ERROR DETECTION

The data to be sent is encapsulated by each layer, from the Application down to the Physical, and each adds it's own header information. When data is received each layer strips off it's header and then passes the packet up to the next layer. The Transport Layer makes sure that the source and destination, hosts and ports, can be identified, and includes a sequence number so that a file can be broken into multiple packets and recombined on the receiving end. The Internet Layer determines how the frames will be delivered, including fragmenting them to send along a path with a smaller maximum transmission unit (MTU) or recombining them for a larger MTU path. It determines the routing used to get to the destination. The Network Layer provides the encapsulation of the datagram into the frame to be transmitted over the network. It includes the ethernet addresses of the source machine and of the next hop towards the destination. These addresses are rewritten with each hop.

11.1 IP HEADER: ---

TABLE 3.

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Version				IHL				TOS								Total length															
Identification																Flags			Fragment offset												
TTL								Protocol								Header checksum															
Source IP address																															
Destination IP address																															
Options and padding :::																															

Version. 4 bits.

Specifies the format of the IP packet header

IHL, Internet Header Length. 4 bits.

Specifies the length of the IP packet header in 32 bit words. The minimum value for a valid header is 5.

TOS, Type of Service. 8 bits.

Specifies the parameters for the type of service requested. The parameters may be utilized by networks to define the handling of the datagram during transport. The M bit was added to this field in.

Precedence. 3 bits.

D. 1 bit.

Minimize delay**T.** 1 bit.
Maximize throughput.. **R.** 1 bit.
Maximize reliability.**M.** 1 bit.
Minimize monetary cost

Total length. 16 bits.

Contains the length of the datagram.

Identification. 16 bits.

Used to identify the fragments of one datagram from those of another. The originating protocol module of an internet datagram sets the identification field to a value that must be unique for that source-destination pair and protocol for the time the datagram will be active in the internet system. The originating protocol module of a complete datagram clears the *MF* bit to zero and the *Fragment Offset* field to zero.

Flags. 3 bits.

R, reserved. 1 bit.

Should be cleared to 0.

DF, Don't fragment. 1 bit.

Controls the fragmentation of the datagram.

MF, More fragments. 1 bit.

Indicates if the datagram contains additional fragments.

Fragment Offset. 13 bits.

Used to direct the reassembly of a fragmented datagram.

TTL, Time to Live. 8 bits.

A timer field used to track the lifetime of the datagram. When the TTL field is decremented down to zero, the datagram is discarded.

Protocol. 8 bits.

This field specifies the next encapsulated protocol

Fragment Offset. 13 bits.

Used to direct the reassembly of a fragmented datagram.

TTL, Time to Live. 8 bits.

A timer field used to track the lifetime of the datagram. When the TTL field is decremented down to zero, the datagram is discarded.

Protocol. 8 bits.

This field specifies the next encapsulated protocol

C, Copy flag. 1 bit.

Indicates if the option is to be copied into all fragments

Class. 2 bits.

Option. 5 bits.

Padding. Variable length Used as a filler to guarantee that the data starts on a 32 bit boundary.

11.2 TCP HEADER:-

TABLE 4.

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Source Port																Destination Port															
Sequence Number																															
Acknowledgment Number																															
Data Offset				Reserved			ECN			Control Bits						Window															
Checksum																Urgent Pointer															
Options and Padding :::																															
Data																															

TCP HEADER FORMAT

TCP segments are sent as internet datagrams. The Internet Protocol header carries several information fields, including the source and destination host addresses [2]. A TCP header follows the Internet header, supplying information specific to the TCP protocol. This division allows for the existence of host level protocols other than TCP.

TCP Header Format

Source Port: 16 bits
 The source port number.

Destination Port: 16 bits
 The destination port number.

Sequence Number: 32 bits
 The sequence number of the first data octet in this segment (except When SYN is present). If SYN is present the sequence number is the Initial sequence number (ISN) and the first data octet is ISN+1.

Acknowledgment Number: 32 bits
 If the ACK control bit is set this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent.

Data Offset: 4 bits
 The number of 32 bit words in the TCP Header. This indicates where the data begins. The TCP header (even one including options) is an integral number of 32 bits long.

Reserved: 6 bits
 Reserved for future use. Must be zero.

Control Bits: 6 bits (from left to right):
 URG: Urgent Pointer field significant
 ACK: Acknowledgment field significant
 PSH: Push Function

RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: No more data from sender

Window: 16 bits

The number of data octets beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept.

Checksum: 16 bits

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16 bit word for checksum purposes. The pad is not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros.

The checksum also covers a 96 bit pseudo header conceptually prefixed to the TCP header. This pseudo header contains the Source Address, the Destination Address, the Protocol, and TCP length. This gives the TCP protection against misrouted segments. This information is carried in the Internet Protocol and is transferred across the TCP/Network interface in the arguments or results of calls by the TCP on the IP.

```
+--------+--------+--------+--------+
|           Source Address          |
+--------+--------+--------+--------+
|         Destination Address       |
+--------+--------+--------+--------+
|  zero  |  PTCL  |    TCP Length   |
+--------+--------+--------+--------+
```

The TCP Length is the TCP header length plus the data length in octets (this is not an explicitly transmitted quantity, but is computed), and it does not count the 12 octets of the pseudo header.

Urgent Pointer: 16 bits

This field communicates the current value of the urgent pointer as a positive offset from the sequence number in this segment. The urgent pointer points to the sequence number of the octet following the urgent data. This field is only be interpreted in segments with the URG control bit set.

Options: variable

Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum. An option may begin on any octet boundary. There are two cases for the format of an option:

Case 1: A single octet of option-kind.
Case 2: An octet of option-kind, an octet of option-length, and
         the actual option-data octets.

The option-length counts the two octets of option-kind and

option-length as well as the option-data octets.
Note that the list of options may be shorter than the data offset field might imply. The content of the header beyond the End-of-Option option must be header padding (i.e., zero).

A TCP must implement all options.
Currently defined options include (kind indicated in octal):

Kind	Length	Meaning
0	-	End of option list.
1	-	No-Operation.
2	4	Maximum Segment Size.

Specific Option Definitions

End of Option List

```
+--------+
|00000000|
+--------+
 Kind=0
```

This option code indicates the end of the option list. This might not coincide with the end of the TCP header according to the Data Offset field. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the TCP header.

No-Operation

```
+--------+
|00000001|
+--------+
 Kind=1
```

This option code may be used between options, for example, to align the beginning of a subsequent option on a word boundary. There is no guarantee that senders will use this option, so receivers must be prepared to process options even if they do not begin on a word boundary.

Maximum Segment Size

```
+--------+--------+---------+--------+
|00000010|00000100|   max seg size   |
+--------+--------+---------+--------+
 Kind=2   Length=4
```

Maximum Segment Size Option Data: 16 bits

If this option is present, then it communicates the maximum receive segment size at the TCP which sends this segment. This field must only be sent in the initial connection request (i.e., in segments with the SYN control bit set). If this option is not used, any segment size is allowed.

Padding: variable

The TCP header padding is used to ensure that the TCP header ends.

Conclusion

In the course of this article, we have examined several Internet-centric firewall designs in an attempt to meet security and performance requirements of multitier applications. In all scenarios, servers hosting application components were separated from the company's corporate network used to conduct internal business, as an initial step to segregate resources with different security requirements. To tightly control interactions between the application's tiers, we looked at hosting tiers of the application on dedicated subnets. By deploying firewalls in series, we were able to significantly increase the difficulty of obtaining unauthorized access to sensitive resources from the Internet. At the same time, each firewall layer increased the design's complexity, contributing to the cost of deploying and maintaining the infrastructure, and increasing the likelihood that it will be misconfigured.

The network design appropriate for your environment depends on the nature of your application and the risks that you are trying to mitigate by setting up a security perimeter around your servers. As we discussed, relying on a single firewall or combining application tiers into a single subnet often decreases the amount of control that you have over how application components are accessed.

However, beware of jumping to a design that incorporates three firewalls in series without first considering less expensive alternatives. In this article, we only touched upon some of the many ways of deploying firewalls with respect to each other, and we did not to examine the relationship between firewalls and other perimeter-defense devices. When designing your network, consider how other components of its perimeter, such as intrusion-detection systems, routers, and VPNs, may impact security of the infrastructure, and select a design that matches your application's architecture and your company's business needs.

PROGRAMMING

```
# INTERNET ACCESS CONTROL SYSTEM DESIGN BASED ON MAC ADDRESS
# WRITTEN AND MAINTAINED BY SAMEER SHARMA ( M.I.T ENGG. COLLEGE
AURANGABAD,M.H)


VERSION=1.A
ECHO -E "\NLOADING VERSION $VERSION............\N"
# WE HAVE TO FIND LOCATION OF IPTABLES
# IF YOUR LINUX DISTRIBUTION CAME WITH A COPY OF IPTABLES, MOST
# LIKELY IT IS LOCATED IN /SBIN.  IF YOU MANUALLY COMPILED
# IPTABLES, THE DEFAULT LOCATION IS IN /USR/LOCAL/SBIN
# TYPE "WHEREIS IPTABLES" COMMAND TO FIND LOCATION OF IPTABLES
# AS REDHAT LINUX-9.0 CAME WITH COPY OF IPTABLES,SO IPTABLES
# LOCATED IN /SBIN DIRECTORY.
ECHO -E "\N ALL DECLERATION.................................\N"
IPTABLES=/SBIN/IPTABLES
LSMOD=/SBIN/LSMOD
GREP=/BIN/GREP
AWK=/BIN/AWK
IFCONFIG=/SBIN/IFCONFIG
DEPMOD=/SBIN/DEPMOD
MODPROBE=/SBIN/MODPROBE
# SETTING THE EXTERNAL AND INTERNAL INTERFACES FOR THE NETWORK

# HERE WE NEED TWO NETWORK ONE FOR EXTERNAL AND ONE INTERNAL NETWORK.
# THE EXTERNAL NETWORK MEANS ETH0 (EXTERNAL NETWORK CARD WHERE ADSL LINE
# CONNECTED) IS WHERE THE NATTING WILL OCCUR AND THE INTERNAL NETWORK
#  SHOULD PREFERABLY BE ADDRESSED WITH A RFC1918 PRIVATE ADDRESS
#  SCHEME.
#  WE USED "ETH0" AS EXTERNAL
#  AND    "ETH1" AS INTERNAL"

#  NOTE:  SOMETIMES CONFIGURATION MAY BE DIFFERENT
#  SO WE HAVE TO MAKE CHANGES AS GIVEN BELOW
#  CHANGE THE EXTIF OR INTIF VARIABLES ABOVE. FOR EXAMPLE:
#  IF YOU ARE A PPPOE OR ANALOG MODEM USER:
#  THAN CHANGE  EXTIF="PPP0"

ECHO -E "\N ALL DECLERATION OF NETWORK........................\N"
ECHO "PLEASE ENTER EXTERNAL INTERFACE "
READ NAME
EXTIF=$NAME
ECHO "PLEASE ENTER INTERNAL INTERFACE "
READ NAME1
INTIF=$NAME1
#LOOIF= "LO"  # SYSTEM NAMES IT..........
INTNET="192.168.1.0/24"
INTIP="192.168.1.1/32"
#LOO_HO="127.0.0.1/32"
#LOC_BC="255.0.0.0"
ANYWHERE="0.0.0.0/0"
```

```
ECHO "  ---"

EXTIP="`$IFCONFIG $EXTIF | $AWK \
/$EXTIF/'{NEXT}//{SPLIT($0,A,":");SPLIT(A[2],A," ");PRINT A[1];EXIT}'`"
ECHO "  EXTERNAL IP: $EXTIP"
ECHO "  ---"
ECHO "  INTERNAL NETWORK: $INTNET"
ECHO "  EXTERNAL INTERFACE:  $EXTIF"
ECHO "  INTERNAL INTERFACE:  $INTIF"
ECHO "  INTERNAL IP:     $INTIP"
ECHO "  LOOP BACK INTERFACE:$LOOIF"
ECHO "  ---"
ECHO "  - VERIFYING THAT ALL KERNEL MODULES ARE OK"
$DEPMOD -A

ECHO -EN "    LOADING KERNEL MODULES: "

ECHO -EN "IP_TABLES, "


#VERIFY THE MODULE ISN'T LOADED.  IF IT IS, SKIP IT
IF [ -Z "` $LSMOD | $GREP IP_TABLES | $AWK {'PRINT $1'} `" ]; THEN
  $MODPROBE IP_TABLES
FI

ECHO -EN "IP_CONNTRACK, "

# LOAD THE STATEFUL CONNECTION TRACKING FRAMEWORK - "IP_CONNTRACK"
# THE CONNTRACK  MODULE IN ITSELF DOES NOTHING WITHOUT OTHER SPECIFIC
# CONNTRACK MODULES BEING LOADED AFTERWARDS SUCH AS THE "IP_CONNTRACK_FTP"
# MODULE
# THIS MODULE IS LOADED AUTOMATICALLY WHEN MASQ FUNCTIONALITY IS
# ENABLED
# LOADED MANUALLY TO CLEAN UP KERNEL AUTO-LOADING TIMING ISSUES

#VERIFY THE MODULE ISN'T LOADED.  IF IT IS, SKIP IT

IF [ -Z "` $LSMOD | $GREP IP_CONNTRACK | $AWK {'PRINT $1'} `" ]; THEN
  $MODPROBE IP_CONNTRACK
FI

ECHO -EN "IP_CONNTRACK_FTP, "

#VERIFY THE MODULE ISN'T LOADED.  IF IT IS, SKIP IT

IF [ -Z "` $LSMOD | $GREP IP_CONNTRACK_FTP | $AWK {'PRINT $1'} `" ]; THEN
  $MODPROBE IP_CONNTRACK_FTP
FI

# LOAD THE GENERAL IPTABLES NAT CODE - "IPTABLE_NAT"
# LOADED AUTOMATICALLY WHEN MASQ FUNCTIONALITY IS TURNED ON
# LOADED MANUALLY TO CLEAN UP KERNEL AUTO-LOADING TIMING ISSUES

#LOAD THE GENERAL IPTABLES NAT CODE - "IPTABLE_NAT"

ECHO -EN "IPTABLE_NAT, "

#VERIFY THE MODULE ISN'T LOADED.  IF IT IS, SKIP IT

IF [ -Z "` $LSMOD | $GREP IPTABLE_NAT | $AWK {'PRINT $1'} `" ]; THEN
  $MODPROBE IPTABLE_NAT
FI
#LOADS THE FTP NAT FUNCTIONALITY INTO THE CORE IPTABLES CODE
# REQUIRED TO SUPPORT NON-PASV FTP.
# ENABLED BY DEFAULT -- INSERT A "#" ON THE NEXT LINE TO DEACTIVATE

ECHO -E "IP_NAT_FTP"
IF [ -Z "` $LSMOD | $GREP IP_NAT_FTP | $AWK {'PRINT $1'} `" ]; THEN
  $MODPROBE IP_NAT_FTP
FI

ECHO "  ---"


ECHO "  ENABLING FORWARDING.."
ECHO "1" > /PROC/SYS/NET/IPV4/IP_FORWARD
ECHO "  ---"
# DEFAULTS CHAINS SETTING ( INPUT, OUTPUT, AND FORWARD) TO DROP

ECHO "  CLEARING ANY EXISTING RULES AND SETTING DEFAULT POLICY TO DROP.."
$IPTABLES -P INPUT DROP
$IPTABLES -F INPUT
$IPTABLES -P OUTPUT DROP
$IPTABLES -F OUTPUT
$IPTABLES -P FORWARD DROP
$IPTABLES -F FORWARD
$IPTABLES -F -T NAT

# DELETE ALL USER-SPECIFIED CHAINS
$IPTABLES -X

# RESET ALL IPTABLES COUNTERS
$IPTABLES -Z
#LATER CHECK AS PER MEMORY.....RIGHT NOW KEEP IT  UP...
#CONFIGURING SPECIFIC CHAINS FOR LATER USE IN THE RULESET

ECHO "  CREATING A DROP CHAIN.."
$IPTABLES -N REJECT-AND-LOG-IT
$IPTABLES -A REJECT-AND-LOG-IT -J LOG --LOG-LEVEL INFO
$IPTABLES -A REJECT-AND-LOG-IT -J REJECT
```

```
########################## IP SPOOFING   ###############################

IP SPOOFING IS A TECHNIQUE USED TO GAIN UNAUTHORIZED ACCESS TO COMPUTERS,
# WHEREBY THE INTRUDER SENDS MESSAGES TO A COMPUTER WITH AN IP ADDRESS
# INDICATING THAT THE MESSAGE IS COMING FROM A TRUSTED HOST. TO ENGAGE
# IN IP SPOOFING, A HACKER MUST FIRST USE A VARIETY OF TECHNIQUES TO FIND
# AN IP ADDRESS OF A TRUSTED HOST AND THEN MODIFY THE PACKET HEADERS SO THAT
# IT APPEARS THAT THE PACKETS ARE COMING FROM THAT HOST.

# HERE THE FOLLOWING LINES ARE USED TO PREVENT THE IP SPOOFING

################## ### PREVENT SPOOF PRIVATE IP #####################

$IPTABLES -A FORWARD -P TCP -S 192.168.1.0/24 -I $EXTIF -J DROP
$IPTABLES -A FORWARD -P UDP -S 192.168.1.0/24 -I $EXTIF -J DROP

ECHO " THIS IS SPOOFING CONTROL SECTION "
ECHO " PREVENT IP SPOOFING IN INPUT TABLE "
$IPTABLES -A INPUT -I $EXTIF -S $INTNET -D $ANYWHERE -J REJECT
ECHO " PREVENT IP SPOOFING IN FORWARD TABLE (PRIVATE IP)"
$IPTABLES -A FORWARD -P TCP -S $INTNET -I $EXTIF -J DROP
$IPTABLES -A FORWARD -P UDP -S $INTNET -I $EXTIF -J DROP
ECHO " PREVENT IP SPOOFING IN NAT PREROUTING TABLE (PRIVATE IP)"
$IPTABLES -T NAT -A PREROUTING -I $EXTIF -S $INTNET -J DROP
$IPTABLES -T NAT -A PREROUTING -I $EXTIF -S 10.0.0.0/8 -J DROP
$IPTABLES -T NAT -A PREROUTING -I $EXTIF -S 172.16.0.0/12 -J DROP
$IPTABLES -T NAT -A PREROUTING -I $EXTIF -S 127.0.0.0/8 -J DROP

# SETTING THE RULES FOR INPUT TABLE OF IPTABLES
ECHO -E "\N   - LOADING INPUT RULESETS"
# LOOPBACK INTERFACES ARE VALID.

$IPTABLES -A INPUT -I LO -S $ANYWHERE -D $ANYWHERE -J ACCEPT

# LOCAL INTERFACE, LOCAL MACHINES, GOING ANYWHERE IS VALID
$IPTABLES -A INPUT -I $INTIF -S $INTNET -D $ANYWHERE -J ACCEPT

$IPTABLES -A INPUT -I $EXTIF -S $INTNET -D $ANYWHERE -J REJECT-AND-LOG-IT

# EXTERNAL INTERFACE, FROM ANY SOURCE, FOR ICMP TRAFFIC IS VALID

#  IF YOU WOULD LIKE YOUR MACHINE TO "PING" FROM THE INTERNET,
#  ENABLE THIS NEXT LINE

#$IPTABLES -A INPUT -I $EXT_IF -P ICMP -S $UNIVERSE -D $EXT_IP -J ACCEPT

# REMOTE INTERFACE, ANY SOURCE, GOING TO THE MASQ SERVERS IP ADDRESS IS VALID
#  STATEFULLY TRACKED

$IPTABLES -A INPUT -I $EXTIF -S $ANYWHERE -D $EXTIP -M STATE --STATE
ESTABLISHED,RELATED -J ACCEPT

$IPTABLES -A INPUT -S $ANYWHERE -D $ANYWHERE -J REJECT-AND-LOG-IT

# SETTING RULES FOR OUTPUT TABLE OF IPTABLES

ECHO -E "   - LOADING OUTPUT RULESETS"
#OUTPUT: OUTGOING TRAFFIC FROM VARIOUS INTERFACES.  ALL RULESETS ARE
#       ALREADY FLUSHED AND SET TO A DEFAULT POLICY OF DROP.
$IPTABLES -A OUTPUT -M STATE -P ICMP --STATE INVALID -J DROP

# LOOPBACK INTERFACE IS VALID.

$IPTABLES -A OUTPUT -O LO -S $ANYWHERE -D $ANYWHERE -J ACCEPT

# LOCAL INTERFACES, ANY SOURCE GOING TO LOCAL NET IS VALID

$IPTABLES -A OUTPUT -O $INTIF -S $EXTIP -D $INTNET -J ACCEPT

# LOCAL INTERFACE, MASQ SERVER SOURCE GOING TO THE LOCAL NET IS VALID

$IPTABLES -A OUTPUT -O $INTIF -S $INTIP -D $INTNET -J ACCEPT

# OUTGOING TO LOCAL NET ON REMOTE INTERFACE, STUFFED ROUTING, DENY

$IPTABLES -A OUTPUT -O $EXTIF -S $ANYWHERE -D $INTNET -J REJECT-AND-LOG-IT

# ANYTHING ELSE OUTGOING ON REMOTE INTERFACE IS VALID

$IPTABLES -A OUTPUT -O $EXTIF -S $EXTIP -D $ANYWHERE -J ACCEPT

$IPTABLES -A OUTPUT -S $ANYWHERE -D $ANYWHERE -J REJECT-AND-LOG-IT

ECHO -E "   - LOADING FORWARD RULESETS"
```

```
##############################################################################

# FORWARD: ENABLE FORWARDING AND THUS IPMASQ
# SETTING THE FORWARD TABLE OF IPTABLES

################# ###########MAIN TESTING ##############################

#######################CONTROLL OVER LOCAL N/W #####################

ECHO "    - FWD: ALLOW ALL CONNECTIONS OUT AND ONLY EXISTING/RELATED IN"
$IPTABLES -A FORWARD -I $EXTIF -O $INTIF -M STATE --STATE
ESTABLISHED,RELATED -J ACCEPT

$IPTABLES -A FORWARD -I $INTIF -O $EXTIF -J ACCEPT

# CATCH ALL RULE, ALL OTHER FORWARDING IS DENIED AND LOGGED.

$IPTABLES -A FORWARD -J REJECT-AND-LOG-IT

##############################################################################

                    ONLY WE HAVE TO DO CHANGES HERE.

##############################################################################

# IN FOLLOWING COMMAND WE HAVE TO PUT CLIENT MAC ADDRESS AND PORT NUMBER
WHICH WE HAVE TO BLOCK............


#$IPTABLES -I FORWARD -P TCP -M MULTIPORT --DESTINATION-PORT (HERE WE HAVE TO
WRITE PORT NUMBERS) -I $INTIF -M MAC --MAC-SOURCE (HERE WE HAVE TO WRITE MAC
ADDRESS) -J REJECT

# FOR EXAMPLE

$IPTABLES -I FORWARD -P TCP -M MULTIPORT --DESTINATION-PORT 80,25 -I $INTIF -M
MAC --MAC-SOURCE 00:03:A1:A0:31:BA -J REJECT
$IPTABLES -I FORWARD -P TCP -M MULTIPORT --DESTINATION-PORT 25 -I $INTIF -M MAC -
-MAC-SOURCE 00:00:00:00:00:01 -J REJECT
$IPTABLES -I FORWARD -P TCP -M MULTIPORT --DESTINATION-PORT 1860,25 -I $INTIF -M
MAC --MAC-SOURCE 00:00:00:00:00:02 -J REJECT
$IPTABLES -I FORWARD -P TCP -M MULTIPORT --DESTINATION-PORT 80 -I $INTIF -M MAC -
-MAC-SOURCE 00:00:00:00:00:03 -J REJECT
$IPTABLES -I FORWARD -P TCP -M MULTIPORT --DESTINATION-PORT 22,21,80,25 -I
$INTIF -M MAC --MAC-SOURCE 00:00:00:00:00:04 -J REJECT

ECHO "    - NAT: ENABLING SNAT (MASQUERADE) FUNCTIONALITY ON $EXT_IF"
```

```
#$IPTABLES -T NAT -A POSTROUTING -O $EXTIF -J MASQUERADE
$IPTABLES -T NAT -A POSTROUTING -O $EXTIF -J SNAT --TO $EXTIP
# INTERNET ACCESS CONTROL SYSTEM DESIGN BASED ON MAC ADDRESS
# WRITTEN AND MAINTAINED BY SAMEER SHARMA ( M.I.T ENGG. COLLEGE
AURANGABAD,M.H)


ECHO -E "\NLOADING VERSION $VERSION...........\N"
# WE HAVE TO FIND LOCATION OF IPTABLES
# IF YOUR LINUX DISTRIBUTION CAME WITH A COPY OF IPTABLES, MOST
# LIKELY IT IS LOCATED IN /SBIN.  IF YOU MANUALLY COMPILED
# IPTABLES, THE DEFAULT LOCATION IS IN /USR/LOCAL/SBIN
# TYPE "WHEREIS IPTABLES" COMMAND TO FIND LOCATION OF IPTABLES
# AS REDHAT LINUX-9.0 CAME WITH COPY OF IPTABLES,SO IPTABLES
# LOCATED IN /SBIN DIRECTORY.
ECHO -E "\N ALL DECLERATION................................\N"
IPTABLES=/SBIN/IPTABLES
LSMOD=/SBIN/LSMOD
GREP=/BIN/GREP
AWK=/BIN/AWK
IFCONFIG=/SBIN/IFCONFIG
DEPMOD=/SBIN/DEPMOD
MODPROBE=/SBIN/MODPROBE
# SETTING THE EXTERNAL AND INTERNAL INTERFACES FOR THE NETWORK

# HERE WE NEED TWO NETWORK ONE FOR EXTERNAL AND ONE INTERNAL NETWORK.
# THE EXTERNAL NETWORK MEANS ETH0 (EXTERNAL NETWORK CARD WHERE ADSL LINE
# CONNECTED) IS WHERE THE NATTING WILL OCCUR AND THE INTERNAL NETWORK
#  SHOULD PREFERABLY BE ADDRESSED WITH A RFC1918 PRIVATE ADDRESS
#  SCHEME.
#  WE USED "ETH0" AS EXTERNAL
#  AND    "ETH1" AS INTERNAL"

#  NOTE:  SOMETIMES CONFIGURATION MAY BE DIFFERENT
#  SO WE HAVE TO MAKE CHANGES AS GIVEN BELOW
#  CHANGE THE EXTIF OR INTIF VARIABLES ABOVE. FOR EXAMPLE:
#  IF YOU ARE A PPPOE OR ANALOG MODEM USER:
#  THAN CHANGE  EXTIF="PPP0"

ECHO -E "\N ALL DECLERATION OF NETWORK.......................\N"
ECHO "PLEASE ENTER EXTERNAL INTERFACE "
READ NAME
EXTIF=$NAME
ECHO "PLEASE ENTER INTERNAL INTERFACE "
READ NAME1
INTIF=$NAME1
#LOOIF= "LO"  # SYSTEM NAMES IT..........
INTNET="192.168.1.0/24"
INTIP="192.168.1.1/32"
#LOO_HO="127.0.0.1/32"
#LOC_BC="255.0.0.0"
```

```
ANYWHERE="0.0.0.0/0"


ECHO "  ---"



EXTIP="`$IFCONFIG $EXTIF | $AWK \
/$EXTIF/'{NEXT}//{SPLIT($0,A,":");SPLIT(A[2],A," ");PRINT A[1];EXIT}'`"
ECHO "  EXTERNAL IP: $EXTIP"
ECHO "  ---"
ECHO "  INTERNAL NETWORK: $INTNET"
ECHO "  EXTERNAL INTERFACE:  $EXTIF"
ECHO "  INTERNAL INTERFACE:  $INTIF"
ECHO "  INTERNAL IP:      $INTIP"
ECHO "  LOOP BACK INTERFACE:$LOOIF"
ECHO "  ---"
ECHO "  - VERIFYING THAT ALL KERNEL MODULES ARE OK"
$DEPMOD -A

ECHO -EN "    LOADING KERNEL MODULES: "

ECHO -EN "IP_TABLES, "


#VERIFY THE MODULE ISN'T LOADED.  IF IT IS, SKIP IT
IF [ -Z "` $LSMOD | $GREP IP_TABLES | $AWK {'PRINT $1'} `" ]; THEN
  $MODPROBE IP_TABLES
FI

ECHO -EN "IP_CONNTRACK, "

# LOAD THE STATEFUL CONNECTION TRACKING FRAMEWORK - "IP_CONNTRACK"
# THE CONNTRACK  MODULE IN ITSELF DOES NOTHING WITHOUT OTHER SPECIFIC
# CONNTRACK MODULES BEING LOADED AFTERWARDS SUCH AS THE "IP_CONNTRACK_FTP"
# MODULE
# THIS MODULE IS LOADED AUTOMATICALLY WHEN MASQ FUNCTIONALITY IS
# ENABLED
# LOADED MANUALLY TO CLEAN UP KERNEL AUTO-LOADING TIMING ISSUES

#VERIFY THE MODULE ISN'T LOADED.  IF IT IS, SKIP IT

IF [ -Z "` $LSMOD | $GREP IP_CONNTRACK | $AWK {'PRINT $1'} `" ]; THEN
  $MODPROBE IP_CONNTRACK
FI

ECHO -EN "IP_CONNTRACK_FTP, "

#VERIFY THE MODULE ISN'T LOADED.  IF IT IS, SKIP IT

IF [ -Z "` $LSMOD | $GREP IP_CONNTRACK_FTP | $AWK {'PRINT $1'} `" ]; THEN
```

```
  $MODPROBE IP_CONNTRACK_FTP
FI

# LOAD THE GENERAL IPTABLES NAT CODE - "IPTABLE_NAT"
# LOADED AUTOMATICALLY WHEN MASQ FUNCTIONALITY IS TURNED ON
# LOADED MANUALLY TO CLEAN UP KERNEL AUTO-LOADING TIMING ISSUES

#LOAD THE GENERAL IPTABLES NAT CODE - "IPTABLE_NAT"

ECHO -EN "IPTABLE_NAT, "

#VERIFY THE MODULE ISN'T LOADED.  IF IT IS, SKIP IT

IF [ -Z "` $LSMOD | $GREP IPTABLE_NAT | $AWK {'PRINT $1'} `" ]; THEN
  $MODPROBE IPTABLE_NAT
FI
#LOADS THE FTP NAT FUNCTIONALITY INTO THE CORE IPTABLES CODE
# REQUIRED TO SUPPORT NON-PASV FTP.
# ENABLED BY DEFAULT -- INSERT A "#" ON THE NEXT LINE TO DEACTIVATE

ECHO -E "IP_NAT_FTP"
IF [ -Z "` $LSMOD | $GREP IP_NAT_FTP | $AWK {'PRINT $1'} `" ]; THEN
  $MODPROBE IP_NAT_FTP
FI

ECHO "  ---"


ECHO "  ENABLING FORWARDING.."
ECHO "1" > /PROC/SYS/NET/IPV4/IP_FORWARD
ECHO "  ---"
# DEFAULTS CHAINS SETTING ( INPUT, OUTPUT, AND FORWARD) TO DROP

ECHO "  CLEARING ANY EXISTING RULES AND SETTING DEFAULT POLICY TO DROP.."
$IPTABLES -P INPUT DROP
$IPTABLES -F INPUT
$IPTABLES -P OUTPUT DROP
$IPTABLES -F OUTPUT
$IPTABLES -P FORWARD DROP
$IPTABLES -F FORWARD
$IPTABLES -F -T NAT

# DELETE ALL USER-SPECIFIED CHAINS
$IPTABLES -X

# RESET ALL IPTABLES COUNTERS
$IPTABLES -Z
#LATER CHECK AS PER MEMORY.....RIGHT NOW KEEP IT  UP...
#CONFIGURING SPECIFIC CHAINS FOR LATER USE IN THE RULESET

ECHO "  CREATING A DROP CHAIN.."
```

```
$IPTABLES -N REJECT-AND-LOG-IT
$IPTABLES -A REJECT-AND-LOG-IT -J LOG --LOG-LEVEL INFO
$IPTABLES -A REJECT-AND-LOG-IT -J REJECT



########################## IP SPOOFING   ###############################

# IP SPOOFING IS A TECHNIQUE USED TO GAIN UNAUTHORIZED ACCESS TO COMPUTERS,
# WHEREBY THE INTRUDER SENDS MESSAGES TO A COMPUTER WITH AN IP ADDRESS
# INDICATING THAT THE MESSAGE IS COMING FROM A TRUSTED HOST. TO ENGAGE
# IN IP SPOOFING, A HACKER MUST FIRST USE A VARIETY OF TECHNIQUES TO FIND
# AN IP ADDRESS OF A TRUSTED HOST AND THEN MODIFY THE PACKET HEADERS SO THAT
# IT APPEARS THAT THE PACKETS ARE COMING FROM THAT HOST.

# HERE THE FOLLOWING LINES ARE USED TO PREVENT THE IP SPOOFING

#########################PREVENT SPOOF PRIVATE IP ####################

$IPTABLES -A FORWARD -P TCP -S 192.168.1.0/24 -I $EXTIF -J DROP
$IPTABLES -A FORWARD -P UDP -S 192.168.1.0/24 -I $EXTIF -J DROP

ECHO " THIS IS SPOOFING CONTROL SECTION "
ECHO " PREVENT IP SPOOFING IN INPUT TABLE "
$IPTABLES -A INPUT -I $EXTIF -S $INTNET -D $ANYWHERE -J REJECT
ECHO " PREVENT IP SPOOFING IN FORWARD TABLE (PRIVATE IP)"
$IPTABLES -A FORWARD -P TCP -S $INTNET -I $EXTIF -J DROP
$IPTABLES -A FORWARD -P UDP -S $INTNET -I $EXTIF -J DROP
ECHO " PREVENT IP SPOOFING IN NAT PREROUTING TABLE (PRIVATE IP)"
$IPTABLES -T NAT -A PREROUTING -I $EXTIF -S $INTNET -J DROP
$IPTABLES -T NAT -A PREROUTING -I $EXTIF -S 10.0.0.0/8 -J DROP
$IPTABLES -T NAT -A PREROUTING -I $EXTIF -S 172.16.0.0/12 -J DROP
$IPTABLES -T NAT -A PREROUTING -I $EXTIF -S 127.0.0.0/8 -J DROP

# SETTING THE RULES FOR INPUT TABLE OF IPTABLES
ECHO -E "\N   - LOADING INPUT RULESETS"
# LOOPBACK INTERFACES ARE VALID.

$IPTABLES -A INPUT -I LO -S $ANYWHERE -D $ANYWHERE -J ACCEPT

# LOCAL INTERFACE, LOCAL MACHINES, GOING ANYWHERE IS VALID
$IPTABLES -A INPUT -I $INTIF -S $INTNET -D $ANYWHERE -J ACCEPT

$IPTABLES -A INPUT -I $EXTIF -S $INTNET -D $ANYWHERE -J REJECT-AND-LOG-IT

# EXTERNAL INTERFACE, FROM ANY SOURCE, FOR ICMP TRAFFIC IS VALID

#  IF YOU WOULD LIKE YOUR MACHINE TO "PING" FROM THE INTERNET,
#  ENABLE THIS NEXT LINE

#$IPTABLES -A INPUT -I $EXT_IF -P ICMP -S $UNIVERSE -D $EXT_IP -J ACCEPT
```

```
# REMOTE INTERFACE, ANY SOURCE, GOING TO THE MASQ SERVERS IP ADDRESS IS VALID

#  STATEFULLY TRACKED

$IPTABLES -A INPUT -I $EXTIF -S $ANYWHERE -D $EXTIP -M STATE --STATE
ESTABLISHED,RELATED -J ACCEPT

$IPTABLES -A INPUT -S $ANYWHERE -D $ANYWHERE -J REJECT-AND-LOG-IT

# SETTING RULES FOR OUTPUT TABLE OF IPTABLES

ECHO -E "   - LOADING OUTPUT RULESETS"
#OUTPUT: OUTGOING TRAFFIC FROM VARIOUS INTERFACES.  ALL RULESETS ARE
#       ALREADY FLUSHED AND SET TO A DEFAULT POLICY OF DROP.
$IPTABLES -A OUTPUT -M STATE -P ICMP --STATE INVALID -J DROP

# LOOPBACK INTERFACE IS VALID.

$IPTABLES -A OUTPUT -O LO -S $ANYWHERE -D $ANYWHERE -J ACCEPT

# LOCAL INTERFACES, ANY SOURCE GOING TO LOCAL NET IS VALID

$IPTABLES -A OUTPUT -O $INTIF -S $EXTIP -D $INTNET -J ACCEPT

# LOCAL INTERFACE, MASQ SERVER SOURCE GOING TO THE LOCAL NET IS VALID

$IPTABLES -A OUTPUT -O $INTIF -S $INTIP -D $INTNET -J ACCEPT

# OUTGOING TO LOCAL NET ON REMOTE INTERFACE, STUFFED ROUTING, DENY

$IPTABLES -A OUTPUT -O $EXTIF -S $ANYWHERE -D $INTNET -J REJECT-AND-LOG-IT

# ANYTHING ELSE OUTGOING ON REMOTE INTERFACE IS VALID

$IPTABLES -A OUTPUT -O $EXTIF -S $EXTIP -D $ANYWHERE -J ACCEPT

$IPTABLES -A OUTPUT -S $ANYWHERE -D $ANYWHERE -J REJECT-AND-LOG-IT

ECHO -E "   - LOADING FORWARD RULESETS"
```

```
############################################################################

# FORWARD: ENABLE FORWARDING AND THUS IPMASQ
# SETTING THE FORWARD TABLE OF IPTABLES


################################   MAIN TESTING ##########################


######################### CONTROLL OVER LOCAL N/W ###################

ECHO "    - FWD: ALLOW ALL CONNECTIONS OUT AND ONLY EXISTING/RELATED IN"
$IPTABLES -A FORWARD -I $EXTIF -O $INTIF -M STATE --STATE
ESTABLISHED,RELATED -J ACCEPT

$IPTABLES -A FORWARD -I $INTIF -O $EXTIF -J ACCEPT

# CATCH ALL RULE, ALL OTHER FORWARDING IS DENIED AND LOGGED.

$IPTABLES -A FORWARD -J REJECT-AND-LOG-IT

############################################################################

                    ONLY WE HAVE TO DO CHANGES HERE.

############################################################################

# IN FOLLOWING COMMAND WE HAVE TO PUT CLIENT MAC ADDRESS AND PORT NUMBER
WHICH WE HAVE TO BLOCK............


#$IPTABLES -I FORWARD -P TCP -M MULTIPORT --DESTINATION-PORT (HERE WE HAVE TO
WRITE PORT NUMBERS) -I $INTIF -M MAC --MAC-SOURCE (HERE WE HAVE TO WRITE MAC
ADDRESS) -J REJECT

# FOR EXAMPLE

$IPTABLES -I FORWARD -P TCP -M MULTIPORT --DESTINATION-PORT 80,25 -I $INTIF -M
MAC --MAC-SOURCE 00:03:A1:A0:31:BA -J REJECT
$IPTABLES -I FORWARD -P TCP -M MULTIPORT --DESTINATION-PORT 25 -I $INTIF -M MAC -
-MAC-SOURCE 00:00:00:00:00:01 -J REJECT
$IPTABLES -I FORWARD -P TCP -M MULTIPORT --DESTINATION-PORT 1860,25 -I $INTIF -M
MAC --MAC-SOURCE 00:00:00:00:00:02 -J REJECT
$IPTABLES -I FORWARD -P TCP -M MULTIPORT --DESTINATION-PORT 80 -I $INTIF -M MAC -
-MAC-SOURCE 00:00:00:00:00:03 -J REJECT
$IPTABLES -I FORWARD -P TCP -M MULTIPORT --DESTINATION-PORT 22,21,80,25 -I
$INTIF -M MAC --MAC-SOURCE 00:00:00:00:00:04 -J REJECT

ECHO "    - NAT: ENABLING SNAT (MASQUERADE) FUNCTIONALITY ON $EXT_IF"

#$IPTABLES -T NAT -A POSTROUTING -O $EXTIF -J MASQUERADE
```

```
$IPTABLES -T NAT -A POSTROUTING -O $EXTIF -J SNAT --TO $EXTIP.
```

```
#!/BIN/SH
CHKCONFIG: 2345 11 89
# WRITTEN AND MAINTAINED BY SAMEER SHARMA
# THIS PROGRAM USED TO RUN FIREWALL NAME AS VIRAT-FIREWALL STORE IN /ETC/RC.D
# DIRECTORY
# LINE AFTER "#" ARE COMMENTS..IT'S ONLY USE TO UNDERSTAND THE PROGRAM
# USAGES:-----
# SERVICE VIRAT-FIREWALL START   " TO RUN FIREWALL"
# SERVICE VIRAT-FIREWALL STOP    " TO STOP FIREWALL"
# SERVICE VIRAT-FIREWALL STATUS  " TO CHECK STATUS OF FIREWALL"
# SERVICE VIRAT-FIREWALL RESTART " TO RESTART THE FIREWALL"

# DESCRIPTION: LOADS THE FIREWALL-IPTABLES RULESET.
# PROCESSNAME: FIREWALL
# PIDFILE: /VAR/RUN/FIREWALL.PID
# CONFIG: /ETC/RC.D/VIRAT-FIREWALL   ( FIREWALL NAME AND PATH )
# PROBE: TRUE

# SOURCE FUNCTION LIBRARY.
. /ETC/RC.D/INIT.D/FUNCTIONS

# CHECK THAT NETWORKING IS UP.
[ "XXXX${NETWORKING}" = "XXXXNO" ] && EXIT 0

[ -X /SBIN/IFCONFIG ] || EXIT 0

#   IF YOUR LINUX DISTRIBUTION CAME WITH A COPY OF IPTABLES, MOST
#   LIKELY IT IS LOCATED IN /SBIN.  IF YOU MANUALLY COMPILED
#   IPTABLES, THE DEFAULT LOCATION IS IN /USR/LOCAL/SBIN
#
#  PLEASE USE THE "WHEREIS IPTABLES" COMMAND TO FIGURE OUT

IPTABLES=/SBIN/IPTABLES


# SEE HOW WE WERE CALLED.
CASE "$1" IN
  START)
    /ETC/RC.D/VIRAT-FIREWALL
    ;;

  STOP)
    ECHO -E "\NFLUSHING FIREWALL AND SETTING DEFAULT POLICIES TO DROP\N"
```

```
   $IPTABLES -P INPUT DROP
   $IPTABLES -F INPUT
   $IPTABLES -P OUTPUT DROP
  $IPTABLES -F OUTPUT
  $IPTABLES -P FORWARD DROP
$IPTABLES -F FORWARD
  $IPTABLES -F -T NAT

   # DELETE ALL USER-SPECIFIED CHAINS
  $IPTABLES -X
   #
   # RESET ALL IPTABLES COUNTERS
 $IPTABLES -Z
   ;;

  RESTART)
      $0 STOP
      $0 START
      ;;

  STATUS)
      $IPTABLES -L
      ;;

  MLIST)
   CAT /PROC/NET/IP_CONNTRACK
   ;;

  *)
      ECHO "USAGE: VIRAT-FIREWALL {START|STOP|STATUS|MLIST}"
      EXIT 1
ESAC

EXIT 0
```

```
/*  AUTHOR: SAMEER SHARMA
*   LAST MODIFIED:2006-MAY-26 10:05:35 AM
*
* DESCRIPTION:
*
* COMPILE WITH:
* GCC  PACKET_CAPTURE_PROGRAM.C -LPCAP
*
* USAGE:
* ./A.OUT (# OF PACKETS) "FILTER STRING"
* MEANS.........
  ./A.OUT 10 "PORT 5050"
     OR
   ./A.OUT -1 "SRC WWW.GOOGLE.COM"
****************************************************************** */



#INCLUDE <PCAP.H>
#INCLUDE <STDIO.H>
#INCLUDE <STDLIB.H>
#INCLUDE <ERRNO.H>
#INCLUDE <SYS/SOCKET.H>
#INCLUDE <NETINET/IN.H>
#INCLUDE <ARPA/INET.H>
#INCLUDE <NETINET/IF_ETHER.H>
#INCLUDE <NET/ETHERNET.H>
#INCLUDE <NETINET/ETHER.H>
#INCLUDE <NETINET/IP.H>
#INCLUDE <NETINET/TCP.H>
//#INCLUDE <FEATURES.H>
/* TCPDUMP HEADER (ETHER.H) DEFINES ETHER_HDRLEN) */
//#DEFINE _BSD_SOURCE 1
//#IFNDEF _FAVOR_BSD
//#DEFINE __FAVOR_BSD 1
//#ENDIF
//#IFNDEF _BSD_SOURCE
//#DEFINE _BSD_SOURCE 1
//#ENDIF
#IFNDEF ETHER_HDRLEN
#DEFINE ETHER_HDRLEN 14
#ENDIF
TYPEDEF U_INT32_T TCP_SEQ;
U_INT16_T HANDLE_ETHERNET(U_CHAR *ARGS,CONST STRUCT PCAP_PKTHDR*
PKTHDR,CONST U_CHAR* PACKET);
      U_CHAR* HANDLE_IP(U_CHAR *ARGS,CONST STRUCT PCAP_PKTHDR* PKTHDR,CONST
U_CHAR* PACKET);
```

```
// ############################################
      U_CHAR* HANDLE_TCP(U_CHAR *ARGS,CONST STRUCT PCAP_PKTHDR* PKTHDR,CONST
U_CHAR* PACKET);
 ########################################################################

STRUCT MY_IP {
            U_INT8_T IP_VHL;                /* HEADER LENGTH, VERSION */
#DEFINE IP_V(IP)   (((IP)->IP_VHL & 0XF0) >> 4)
#DEFINE IP_HL(IP) ((IP)->IP_VHL & 0X0F)
            U_INT8_T IP_TOS;                /* TYPE OF SERVICE */
            U_INT16_T           IP_LEN;                 /* TOTAL LENGTH */
            U_INT16_T           IP_ID;                  /* IDENTIFICATION */
            U_INT16_T           IP_OFF;                 /* FRAGMENT OFFSET FIELD */
#DEFINE  IP_RF 0X8000                               /* DONT FRAGMENT FLAG */
#DEFINE  IP_DF 0X4000                               /* DONT FRAGMENT FLAG */
#DEFINE  IP_MF 0X2000                               /* MORE FRAGMENTS FLAG */
#DEFINE  IP_OFFMASK 0X1FFF                          /* MASK FOR FRAGMENTING BITS */
            U_INT8_T IP_TTL;                /* TIME TO LIVE */
            U_INT8_T IP_P;                  /* PROTOCOL */
            U_INT16_T           IP_SUM;                 /* CHECKSUM */
            STRUCT   IN_ADDR IP_SRC,IP_DST;         /* SOURCE AND DEST ADDRESS */
};
//    #####################################################################

STRUCT MY_TCP
 {

   U_INT16_T TH_SPORT;
   U_INT16_T TH_DPORT;
  // U_SHORT TH_SPORT;
  // U_SHORT TH_DPORT;
   TCP_SEQ TH_SEQ;
   TCP_SEQ TH_ACK;
#  IF __BYTE_ORDER == __LITTLE_ENDIAN
   U_INT TH_X2:4;
   U_INT TH_OFF:4;
#  ENDIF
#  IF __BYTE_ORDER == __BIG_ENDIAN
   U_INT TH_OFF:4;
   U_INT TH_X2:4;
#  ENDIF
   U_CHAR TH_FLAGS;
#  DEFINE TH_FIN      0X01
#  DEFINE TH_SYN      0X02
#  DEFINE TH_RST      0X04
#  DEFINE TH_PUSH     0X08
#  DEFINE TH_ACK      0X10
#  DEFINE TH_URG      0X20
#  DEFINE TH_ECE      0X40
#  DEFINE TH_CWR      0X80
# DEFINE TH_FLAGS
(TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
   U_SHORT TH_WIN;
   U_SHORT TH_SUM;
   U_SHORT TH_URP;
 };

VOID MY_CALLBACK(U_CHAR *ARGS,CONST STRUCT PCAP_PKTHDR* PKTHDR,CONST
U_CHAR* PACKET)
{
 //#########################################################
   HANDLE_TCP(ARGS,PKTHDR,PACKET);
  // ########################################################
   U_INT16_T TYPE = HANDLE_ETHERNET(ARGS,PKTHDR,PACKET);
   IF(TYPE == ETHERTYPE_IP)
   {
             /* HANDLE IP PACKET */
      HANDLE_IP(ARGS,PKTHDR,PACKET);
   }
   ELSE IF(TYPE == ETHERTYPE_ARP)
   {
     /* HANDLE ARP PACKET */
   }
   ELSE IF
   (TYPE == ETHERTYPE_REVARP)
   {/* HANDLE REVERSE ARP PACKET */
   }
}

U_INT16_T HANDLE_ETHERNET(U_CHAR *ARGS,CONST STRUCT PCAP_PKTHDR*
PKTHDR,CONST U_CHAR* PACKET)
{
   U_INT CAPLEN = PKTHDR->CAPLEN;
   U_INT LENGTH = PKTHDR->LEN;
   STRUCT ETHER_HEADER *EPTR;  /* NET/ETHERNET.H */

       INT SIZE_ETH = SIZEOF(STRUCT ETHER_HEADER);
      PRINTF("\N SIZE OF ETHERNET%D\N\N\N\N",SIZE_ETH);


// LATER WE VE TO SEE IT......................
   STRUCT PCAP_PKTHDR HDR;
   U_SHORT ETHER_TYPE;

   IF (CAPLEN < ETHER_HDRLEN)
   {
      FPRINTF(STDOUT,"PACKET LENGTH LESS THAN ETHERNET HEADER LENGTH\N");
      RETURN -1;
   }

   /* LETS START WITH THE ETHER HEADER... */
   EPTR = (STRUCT ETHER_HEADER *)(PACKET);
   ETHER_TYPE = NTOHS(EPTR->ETHER_TYPE);
   /* LETS PRINT SOURCE DEST TYPE LENGTH */
//WE WILL SEE LATER    FPRINTF(STDOUT,"SOURCE ETHERNET ADDRESS      : ");
   FPRINTF(STDOUT,"DESTINATION ETHERNET ADDRESS : ");
   FPRINTF(STDOUT,"%S\N",ETHER_NTOA((STRUCT ETHER_ADDR*)EPTR->ETHER_SHOST));
// WE WILL SEE LATER    FPRINTF(STDOUT,"DESTINATION ETHERNET ADDRESS : ");
   FPRINTF(STDOUT,"SOURCE ETHERNET ADDRESS      : ");
   FPRINTF(STDOUT,"%S\N",ETHER_NTOA((STRUCT ETHER_ADDR*)EPTR->ETHER_DHOST));
   /* CHECK TO SEE IF WE HAVE AN IP PACKET */
   IF (ETHER_TYPE == ETHERTYPE_IP)
   {
      FPRINTF(STDOUT,"\NTYPE(IP)CAPTURED PACKET LENGTH  : ");
   }
   ELSE  IF (ETHER_TYPE == ETHERTYPE_ARP)
   {
      FPRINTF(STDOUT,"\NTYPE(ARP)CAPTURED PACKET LENGTH : ");
   }
   ELSE  IF (EPTR->ETHER_TYPE == ETHERTYPE_REVARP)
   {
  FPRINTF(STDOUT,"\NTYPE(RARP)CAPTURED PACKET LENGTH: ");
   }
   ELSE
   {
      FPRINTF(STDOUT,"(?)");
   }
   FPRINTF(STDOUT," %D\N",LENGTH);
   PRINTF("RECIEVED AT........ %S\N",CTIME((CONST TIME_T*)&HDR.TS.TV_SEC));
   PRINTF("ETHERNET ADDRESS LENGTH IS      : %D\N",ETHER_HDR_LEN);
   RETURN ETHER_TYPE;
}

U_CHAR* HANDLE_IP(U_CHAR *ARGS,CONST STRUCT PCAP_PKTHDR* PKTHDR,CONST
U_CHAR* PACKET)
{
   CONST STRUCT MY_IP* IP;
   U_INT LENGTH = PKTHDR->LEN;
   U_INT HLEN,OFF,VERSION;
   INT I;
   INT LEN;
// TESTING........
              INT SIZE_IP = SIZEOF(STRUCT MY_IP);
       PRINTF("\N\N\NSIZE OF IP \N\%D\N\N\N\N",SIZE_IP);
      INT ETHER1;
             ETHER1=SIZEOF(STRUCT ETHER_HEADER);
//           PRINTF("\NETHER============%D\N",ETHER1);
   /* JUMP PASS THE ETHERNET HEADER */
   IP = (STRUCT MY_IP*)(PACKET + ETHER1);
//  IP = (STRUCT MY_IP*)(PACKET + SIZEOF(STRUCT ETHER_HEADER));

//   LENGTH -= (SIZEOF(STRUCT ETHER_HEADER)+SIZEOF(STRUCT MY_TCP));
```

```
   LENGTH -= SIZEOF(STRUCT ETHER_HEADER);
   /* CHECK TO SEE WE HAVE A PACKET OF VALID LENGTH */
//   IF (LENGTH < SIZEOF(STRUCT MY_IP))
//   {
//      PRINTF("TRUNCATED IP %D \N",LENGTH);
//      RETURN NULL;
//   }

   LEN = NTOHS(IP->IP_LEN);
   HLEN = IP_HL(IP); /* HEADER LENGTH */
   VERSION = IP_V(IP);/* IP VERSION */
   /* CHECK VERSION */
   IF(VERSION != 4)
   {
     FPRINTF(STDOUT,"UNKNOWN VERSION %D\N",VERSION);
     RETURN NULL;
   }

   /* CHECK HEADER LENGTH */
   IF(HLEN < 5 )
   {
      FPRINTF(STDOUT,"BAD-HLEN %D\N",HLEN);
   }

   /* SEE IF WE HAVE AS MUCH PACKET AS WE SHOULD */
   IF(LENGTH < LEN)
      PRINTF("\NTRUNCATED IP - %D BYTES MISSING\N",LEN - LENGTH);
   /* CHECK TO SEE IF WE HAVE THE FIRST FRAGMENT */
   OFF = NTOHS(IP->IP_OFF);
   IF((OFF & 0X1FFF) == 0 )/* AKA NO 1'S IN FIRST 13 BITS */
   {/* PRINT SOURCE DESTINATION HLEN VERSION LEN OFFSET */
// LATER WE WILL SEE        FPRINTF(STDOUT,"SOURCE IP ADDRESS     : " );
      FPRINTF(STDOUT,"DESTINATION IP ADDRESS : " );
      FPRINTF(STDOUT,"%S\N",INET_NTOA(IP->IP_SRC));
// LATER WE WILL SEE       FPRINTF(STDOUT,"DESTINATION IP ADDRESS : " );
      FPRINTF(STDOUT,"SOURCE IP ADDRESS     : " );
             FPRINTF(STDOUT,"%S\N\N",INET_NTOA(IP->IP_DST));
    //  FPRINTF(STDOUT,"%D\N%D\N%D\N%D\N",HLEN,VERSION,LEN,OFF);
      FPRINTF(STDOUT,"FRAGMENT OFFSET LENGTH :%D\N",OFF);
      FPRINTF(STDOUT,"LENGTH OF THE IP PACKET:%D\N",LEN);
      FPRINTF(STDOUT,"VERSION OF THE   PACKET:%D\N",VERSION);
      FPRINTF(STDOUT,"HEADER LENGTH OF PACKET:%D\N",HLEN);
     }
   RETURN NULL;
}
// ####################################################


U_CHAR* HANDLE_TCP(U_CHAR *ARGS,CONST STRUCT PCAP_PKTHDR* PKTHDR,CONST
U_CHAR* PACKET)
{
        CONST STRUCT MY_TCP* TCP;
        U_INT LENGTH = PKTHDR->LEN;
 //       U_INT16_T SPORT=0;
 //     U_INT16_T DPORT=0;
        U_SHORT SPORT=0;
        U_SHORT DPORT=0;
        TCP_SEQ SEQ;
        TCP_SEQ ACK;
        SEQ=TCP->TH_SEQ;
        ACK=TCP->TH_ACK;
        INT WIN,SUM,URP;
        WIN=TCP->TH_WIN;
        SUM=TCP->TH_SUM;
        URP=TCP->TH_URP;
//      INT SPORT=0,DPORT=0;
        INT SIZE_TCP = SIZEOF(STRUCT MY_TCP);
        PRINTF(" CAPTURED PACKET OF LENGTH %D",LENGTH);
        PRINTF("\N\N\NSIZE OF TCP \N\%D\N\N\N\N",SIZE_TCP);
       TCP = (STRUCT MY_TCP*)(PACKET+34);
        PRINTF("\N\N\N\NTCP HEADER IS %D",LENGTH);
        SPORT=NTOHS(TCP->TH_SPORT);//13056
        DPORT=NTOHS(TCP->TH_DPORT);//44 28 VARRYING.......TCP 33
        WIN=NTOHS(TCP->TH_WIN);
        SUM=NTOHS(TCP->TH_SUM);
        URP=NTOHS(TCP->TH_URP);
//              SPORT=TCP->TH_SPORT;//SRC 51
//              DPORT=TCP->TH_DPORT;// 40912 VARRYING
        FPRINTF(STDOUT,"SOURCE PORT: %D\N",SPORT);
        FPRINTF(STDOUT,"DESTINATION PORT: %D\N",DPORT);
        FPRINTF(STDOUT,"SEQUENCE NO.: %D\N",SEQ);
        FPRINTF(STDOUT,"SEQUENCE NO.: %D\N",ACK);
        FPRINTF(STDOUT,"SUM.: %D\N",SUM);
        FPRINTF(STDOUT,"WIN.: %D\N",WIN);
        FPRINTF(STDOUT,"URP : %D\N",URP);
        RETURN NULL;
}
```

```
// ############################################################

INT MAIN(INT ARGC,CHAR **ARGV)
{
   CHAR *DEV;
   CHAR *MASK;
   CHAR ERRBUF[PCAP_ERRBUF_SIZE];
   PCAP_T* DESCR;
   STRUCT BPF_PROGRAM FP;      /* HOLD COMPILED PROGRAM     */
   BPF_U_INT32 MASKP;          /* SUBNET MASK              */
   BPF_U_INT32 NETP;           /* IP                       */
   STRUCT IN_ADDR ADDR;
   INT RESULT;
   U_CHAR* ARGS = NULL;

  /* OPTIONS MUST BE PASSED IN AS A STRING BECAUSE I AM LAZY */
   IF(ARGC < 2){
      FPRINTF(STDOUT,"USAGE: %S NUMPACKETS \"OPTIONS\"\N",ARGV[0]);
      RETURN 0;
   }
   /* GRAB A DEVICE TO PEAK INTO... */
   DEV = PCAP_LOOKUPDEV(ERRBUF);
   PRINTF("DEVICE NAME  : %S\N",DEV);
   IF(DEV == NULL)
   {
   PRINTF("%S\N",ERRBUF);
   EXIT(1);
   }
   /* ASK PCAP FOR THE NETWORK ADDRESS AND MASK OF THE DEVICE */
   RESULT=PCAP_LOOKUPNET(DEV,&NETP,&MASKP,ERRBUF);
   IF(RESULT == -1)
   {
   PRINTF("%S\N",ERRBUF);
   EXIT(1);
   }
   ADDR.S_ADDR = MASKP;
   MASK = INET_NTOA(ADDR);

   IF(MASK == NULL)
   {
   PERROR("INET_NTOA");
   EXIT(1);
   }
   PRINTF("NETWORK MASK : %S\N\N",MASK);

   /* OPEN DEVICE FOR READING. NOTE: DEFAULTING TO
    * PROMISCUOUS MODE*/
   DESCR = PCAP_OPEN_LIVE("ETH0",BUFSIZ,1,-1,ERRBUF);
   IF(DESCR == NULL)
   {
   PRINTF("PCAP_OPEN_LIVE(): %S\N",ERRBUF);
```

```
    EXIT(1);
    }
    IF(ARGC > 2)
    {
       /* LETS TRY AND COMPILE THE PROGRAM.. NON-OPTIMIZED */
       IF(PCAP_COMPILE(DESCR,&FP,ARGV[2],0,NETP) == -1)
       {
       FPRINTF(STDERR,"ERROR CALLING PCAP_COMPILE\N");
       EXIT(1);
       }
       /* SET THE COMPILED PROGRAM AS THE FILTER */
       IF(PCAP_SETFILTER(DESCR,&FP) == -1)
       {
       FPRINTF(STDERR,"ERROR SETTING FILTER\N");
       EXIT(1);
       }
    }
    /* ... AND LOOP */
    PCAP_LOOP(DESCR,ATOI(ARGV[1]),MY_CALLBACK,ARGS);
    FPRINTF(STDOUT,"\NTHE PACKET CAPTURING OPERATION IS COMPLETED\NFINISHED\N");
    RETURN 0;
}
```

PCAP LIBRARY: ---

THE PACKET CAPTURE LIBRARY PROVIDES A HIGH LEVEL INTERFACE TO PACKET CAPTURE SYSTEMS. ALL PACKETS ON THE NETWORK, EVEN THOSE DESTINED FOR OTHER HOSTS, ARE ACCESSIBLE THROUGH THIS MECHANISM.

PCAP - PACKET CAPTURE LIBRARY

```
#include <pcap.h>
char errbuf[PCAP_ERRBUF_SIZE];
pcap_t *pcap_open_live(const char *device, int snaplen,
        int promisc, int to_ms, char *errbuf)
pcap_t *pcap_open_dead(int linktype, int snaplen)
pcap_t *pcap_open_offline(const char *fname, char *errbuf)
pcap_dumper_t *pcap_dump_open(pcap_t *p, const char *fname)
int pcap_setnonblock(pcap_t *p, int nonblock, char *errbuf);
int pcap_getnonblock(pcap_t *p, char *errbuf);
int pcap_findalldevs(pcap_if_t **alldevsp, char *errbuf)
void pcap_freealldevs(pcap_if_t *alldevs)
char *pcap_lookupdev(char *errbuf)
int pcap_lookupnet(const char *device, bpf_u_int32 *netp,
        bpf_u_int32 *maskp, char *errbuf)
int pcap_dispatch(pcap_t *p, int cnt,


        pcap_handler callback, u_char *user)
int pcap_loop(pcap_t *p, int cnt,
        pcap_handler callback, u_char *user)
void pcap_dump(u_char *user, struct pcap_pkthdr *h,
      u_char *sp)
int pcap_compile(pcap_t *p, struct bpf_program *fp,
     char *str, int optimize, bpf_u_int32 netmask)
int pcap_setfilter(pcap_t *p, struct bpf_program *fp)
void pcap_freecode(struct bpf_program *);
const u_char *pcap_next(pcap_t *p, struct pcap_pkthdr *h)
int pcap_next_ex(pcap_t *p, struct pcap_pkthdr **pkt_header,
 const u_char **pkt_data)
void pcap_breakloop(pcap_t *)
int pcap_datalink(pcap_t *p)
int pcap_list_datalinks(pcap_t *p, int **dlt_buf);
int pcap_set_datalink(pcap_t *p, int dlt);
int pcap_datalink_name_to_val(const char *name);
const char *pcap_datalink_val_to_name(int dlt);
const char *pcap_datalink_val_to_description(int dlt);
int pcap_snapshot(pcap_t *p)
int pcap_is_swapped(pcap_t *p)
int pcap_major_version(pcap_t *p)
int pcap_minor_version(pcap_t *p)
int pcap_stats(pcap_t *p, struct pcap_stat *ps)
FILE *pcap_file(pcap_t *p)
int pcap_fileno(pcap_t *p)
void pcap_perror(pcap_t *p, char *prefix)
char *pcap_geterr(pcap_t *p)
char *pcap_strerror(int error)
const char *pcap_lib_version(void)
void pcap_close(pcap_t *p)
int pcap_dump_flush(pcap_dumper_t *p)
FILE *pcap_dump_file(pcap_dumper_t *p)
void pcap_dump_close(pcap_dumper_t *p)
```

pcap_open_live() is used to obtain a packet capture descriptor to look at packets on the network. *device* is a string that specifies the network device to open; on Linux systems with 2.2 or later kernels, a *device* argument of "any" or **NULL** can be used to capture packets from all interfaces. *snaplen* specifies the maximum number of bytes to capture. If this value is less than the size of a packet that is captured, only the first *snaplen* bytes of that packet will be captured and provided as packet data. A value of 65535 should be sufficient, on most if not all networks, to capture all the data available from the packet. *promisc* specifies if the interface is to be put into promiscuous mode. (Note that even if this parameter is false, the interface could well be in promiscuous mode for some other reason.) For now, this doesn't work on the "any" device; if an argument of "any" or NULL is supplied, the *promisc* flag is ignored. *to_ms* specifies the read timeout in milliseconds. The read timeout is used to arrange that the read not necessarily return immediately when a packet is seen, but that it wait for some amount of time to allow more packets to arrive and to read multiple packets from the OS kernel in one operation. Not all platforms support a read timeout; on platforms that don't, the read timeout is ignored. A zero value for *to_ms*, on platforms that support a read timeout, will cause a read to wait forever to allow enough packets to arrive, with no timeout. *errbuf* is used to return error or warning text. It will be set to error text when **pcap_open_live()** fails and returns **NULL**. *errbuf* may also be set to warning text when **pcap_open_live()** succeds; to detect this case the caller should store a zero-length string in *errbuf* before calling **pcap_open_live()** and display the warning to the user if *errbuf* is no longer a zero-length string.
pcap_open_dead() is used for creating a **pcap_t** structure to use when calling the other functions in libpcap. It is typically used when just using libpcap for compiling BPF code.
pcap_open_offline() is called to open a ``savefile'' for reading. *fname* specifies the name of the file to open. The file has the same format as those used by **tcpdump(1)** and **tcpslice(1)**. The name "-" in a synonym for **stdin**. *errbuf* is used to return error text and is only set when **pcap_open_offline()** fails and returns **NULL**.
pcap_dump_open() is called to open a ``savefile'' for writing. The name "-" in a synonym for **stdout**. **NULL** is returned on failure. *p* is a *pcap* struct as returned by **pcap_open_offline()** or **pcap_open_live()**. *fname* specifies the name of the file to open. If **NULL** is returned, **pcap_geterr()** can be used to get the error text.
pcap_setnonblock() puts a capture descriptor, opened with **pcap_open_live()**, into ``non-blocking'' mode, or takes it out of ``non-blocking'' mode, depending on whether the *nonblock* argument is non-zero or zero. It has no effect on ``savefiles''. If there is an error, -1 is returned and *errbuf* is filled in with an appropriate error message; otherwise, 0 is returned. In ``non-blocking'' mode, an attempt to read from the capture descriptor with **pcap_dispatch()** will, if no packets are currently available to be read, return 0 immediately rather than blocking waiting for packets to arrive. **pcap_loop()** and **pcap_next()** will not work in ``non-blocking'' mode.
pcap_getnonblock() returns the current ``non-blocking'' state of the capture descriptor; it always returns 0 on ``savefiles''. If there is an error, -1 is returned and *errbuf* is filled in with an appropriate error message.
pcap_findalldevs() constructs a list of network devices that can be opened with **pcap_open_live()**. (Note that there may be network devices that cannot be opened with **pcap_open_live()** by the process calling **pcap_findalldevs()**, because, for example, that process might not have sufficient privileges to open them for capturing; if so, those devices will not appear on the list.) *alldevsp* is set to point to the first element of the list; each element of the list is of type **pcap_if_t**, and has the following members:

> **NEXT** IF NOT **NULL**, A POINTER TO THE NEXT ELEMENT IN THE LIST; **NULL** FOR THE LAST ELEMENT OF THE LIST

NAME A POINTER TO A STRING GIVING A NAME FOR THE DEVICE TO PASS TO **PCAP_OPEN_LIVE() DESCRIPTION**
IF NOT **NULL**, A POINTER TO A STRING GIVING A HUMAN-READABLE DESCRIPTION OF THE DEVICE
ADDRESSES A POINTER TO THE FIRST ELEMENT OF A LIST OF ADDRESSES FOR THE INTERFACE
FLAGS INTERFACE FLAGS:
PCAP_IF_LOOPBACK SET IF THE INTERFACE IS A LOOPBACK INTERFACE

Each element of the list of addresses is of type **pcap_addr_t**, and has the following members:

NEXT IF NOT **NULL**, A POINTER TO THE NEXT ELEMENT IN THE LIST; **NULL** FOR THE LAST ELEMENT OF THE LIST
ADDR A POINTER TO A **STRUCT SOCKADDR** CONTAINING AN ADDRESS
NETMASK IF NOT **NULL**, A POINTER TO A **STRUCT SOCKADDR** THAT CONTAINS THE NETMASK CORRESPONDING TO THE ADDRESS POINTED TO BY **ADDR**
BROADADDR IF NOT **NULL**, A POINTER TO A **STRUCT SOCKADDR** THAT CONTAINS THE BROADCAST ADDRESS CORRESPONDING TO THE ADDRESS POINTED TO BY **ADDR**; MAY BE NULL IF THE INTERFACE DOESN'T SUPPORT BROADCASTS
DSTADDR IF NOT **NULL**, A POINTER TO A **STRUCT SOCKADDR** THAT CONTAINS THE DESTINATION ADDRESS CORRESPONDING TO THE ADDRESS POINTED TO BY **ADDR**; MAY BE NULL IF THE INTERFACE ISN'T A POINT-TO-POINT INTERFACE

-1 is returned on failure, in which case **errbuf** is filled in with an appropriate error message; **0** is returned on success.
pcap_freealldevs() is used to free a list allocated by **pcap_findalldevs()**.
pcap_lookupdev() returns a pointer to a network device suitable for use with **pcap_open_live()** and **pcap_lookupnet()**. If there is an error, **NULL** is returned and *errbuf* is filled in with an appropriate error message.
pcap_lookupnet() is used to determine the network number and mask associated with the network device **device**. Both *netp* and *maskp* are *bpf_u_int32* pointers. A return of -1 indicates an error in which case *errbuf* is filled in with an appropriate error message.
pcap_dispatch() is used to collect and process packets. *cnt* specifies the maximum number of packets to process before returning. This is not a minimum number; when reading a live capture, only one bufferful of packets is read at a time, so fewer than *cnt* packets may be processed. A *cnt* of -1 processes all the packets received in one buffer when reading a live capture, or all the packets in the file when reading a ``savefile''. *callback* specifies a routine to be called with three arguments: a *u_char* pointer which is passed in from **pcap_dispatch()**, a *const struct pcap_pkthdr* pointer to a structure with the following members:

TS A *STRUCT TIMEVAL* CONTAINING THE TIME WHEN THE PACKET WAS CAPTURED
CAPLEN A *BPF_U_INT32* GIVING THE NUMBER OF BYTES OF THE PACKET THAT ARE AVAILABLE FROM THE CAPTURE
LEN A *BPF_U_INT32* GIVING THE LENGTH OF THE PACKET, IN BYTES (WHICH MIGHT BE MORE THAN THE NUMBER OF BYTES AVAILABLE FROM THE CAPTURE, IF THE LENGTH OF THE PACKET IS LARGER THAN THE MAXIMUM NUMBER OF BYTES TO CAPTURE)

and a *const u_char* pointer to the first **caplen** (as given in the *struct pcap_pkthdr* a pointer to which is passed to the callback routine) bytes of data from the packet (which won't necessarily be the entire packet; to capture the entire packet, you will have to provide a value for *snaplen* in your call to **pcap_open_live()** that is sufficiently large to get all of the packet's data - a value of 65535 should be sufficient on most if not all networks).

The number of packets read is returned. 0 is returned if no packets were read from a live capture (if, for example, they were discarded because they didn't pass the packet filter, or if, on platforms that support a read timeout that starts before any packets arrive, the timeout expires before any packets arrive, or if the file descriptor for the capture device is in non-blocking mode and no packets were available to be read) or if no more packets are available in a ``savefile.'' A return of -1 indicates an error in which case **pcap_perror()** or **pcap_geterr()** may be used to display the error text. A return of -2 indicates that the loop terminated due to a call to **pcap_breakloop()** before any packets were processed. **If your application uses pcap_breakloop(), make sure that you explicitly check for -1 and -2, rather than just checking for a return value < 0.**
NOTE: when reading a live capture, **pcap_dispatch()** will not necessarily return when the read times out; on some platforms, the read timeout isn't supported, and, on other platforms, the timer doesn't start until at least one packet arrives. This means that the read timeout should **NOT** be used in, for example, an interactive application, to allow the packet capture loop to ``poll'' for user input periodically, as there's no guarantee that **pcap_dispatch()** will return after the timeout expires.
pcap_loop() is similar to **pcap_dispatch()** except it keeps reading packets until *cnt* packets are processed or an error occurs. It does **not** return when live read timeouts occur. Rather, specifying a non-zero read timeout to **pcap_open_live()** and then calling **pcap_dispatch()** allows the reception and processing of any packets that arrive when the timeout occurs. A negative *cnt* causes **pcap_loop()** to loop forever (or at least until an error occurs). -1 is returned on an error; 0 is returned if *cnt* is exhausted; -2 is returned if the loop terminated due to a call to **pcap_breakloop()** before any packets were processed. **If your application uses pcap_breakloop(), make sure that you explicitly check for -1 and -2, rather than just checking for a return value < 0.**
pcap_next() reads the next packet (by calling **pcap_dispatch()** with a *cnt* of 1) and returns a *u_char* pointer to the data in that packet. (The *pcap_pkthdr* struct for that packet is not supplied.) **NULL** is returned if an error occured, or if no packets were read from a live capture (if, for example, they were discarded because they didn't pass the packet filter, or if, on platforms that support a read timeout that starts before any packets arrive, the timeout expires before any packets arrive, or if the file descriptor for the capture device is in non-blocking mode and no packets were available to be read), or if no more packets are available in a ``savefile.'' Unfortunately, there is no way to determine whether an error occured or not. **pcap_next_ex()** reads the next packet and returns a success/failure indication:

1 THE PACKET WAS READ WITHOUT PROBLEMS
0 PACKETS ARE BEING READ FROM A LIVE CAPTURE, AND THE TIMEOUT EXPIRED
-1 AN ERROR OCCURRED WHILE READING THE PACKET
-2 PACKETS ARE BEING READ FROM A ``SAVEFILE'', AND THERE ARE NO MORE PACKETS TO READ FROM THE SAVEFILE.

If the packet was read without problems, the pointer pointed to by the *pkt_header* argument is set to point to the *pcap_pkthdr* struct for the packet, and the pointer pointed to by the *pkt_data* argument is set to point to the data in the packet.
pcap_breakloop() sets a flag that will force **pcap_dispatch()** or **pcap_loop()** to return rather than looping; they will return the number of packets that have been processed so far, or -2 if no packets have been processed so far.
This routine is safe to use inside a signal handler on UNIX or a console control handler on Windows, as it merely sets a flag that is checked within the loop.
The flag is checked in loops reading packets from the OS - a signal by itself will not necessarily terminate those loops - as well as in loops processing a set of packets returned by the OS. Note that if you are catching signals on UNIX systems that support

restarting system calls after a signal, and calling pcap_breakloop() in the signal handler, you must specify, when catching those signals, that system calls should NOT be restarted by that signal. Otherwise, if the signal interrupted a call reading packets in a live capture, when your signal handler returns after calling pcap_breakloop(), the call will be restarted, and the loop will not terminate until more packets arrive and the call completes**.**

Note that **pcap_next()** will, on some platforms, loop reading packets from the OS; that loop will not necessarily be terminated by a signal, so **pcap_breakloop()** should be used to terminate packet processing even if **pcap_next()** is being used.

pcap_breakloop() does not guarantee that no further packets will be processed by **pcap_dispatch()** or **pcap_loop()** after it is called; at most one more packet might be processed.

If -2 is returned from **pcap_dispatch()** or **pcap_loop()**, the flag is cleared, so a subsequent call will resume reading packets. If a positive number is returned, the flag is not cleared, so a subsequent call will return -2 and clear the flag.

pcap_dump() outputs a packet to the ``savefile'' opened with **pcap_dump_open()**. Note that its calling arguments are suitable for use with **pcap_dispatch()** or **pcap_loop()**. If called directly, the *user* parameter is of type *pcap_dumper_t* as returned by **pcap_dump_open()**.

pcap_compile() is used to compile the string *str* into a filter program. *program* is a pointer to a *bpf_program* struct and is filled in by **pcap_compile()**. *optimize* controls whether optimization on the resulting code is performed. *netmask* specifies the IPv4 netmask of the network on which packets are being captured; it is used only when checking for IPv4 broadcast addresses in the filter program. If the netmask of the network on which packets are being captured isn't known to the program, or if packets are being captured on the Linux "any" pseudo-interface that can capture on more than one network, a value of 0 can be supplied; tests for IPv4 broadcast addreses won't be done correctly, but all other tests in the filter program will be OK. A return of -1 indicates an error in which case **pcap_geterr()** may be used to display the error text.

pcap_compile_nopcap() is similar to **pcap_compile()** except that instead of passing a pcap structure, one passes the snaplen and linktype explicitly. It is intended to be used for compiling filters for direct BPF usage, without necessarily having called **pcap_open()**. A return of -1 indicates an error; the error text is unavailable. (**pcap_compile_nopcap()** is a wrapper around **pcap_open_dead()**, **pcap_compile()**, and **pcap_close()**; the latter three routines can be used directly in order to get the error text for a compilation error.)

pcap_setfilter() is used to specify a filter program. *fp* is a pointer to a *bpf_program* struct, usually the result of a call to **pcap_compile()**. **-1** is returned on failure, in which case **pcap_geterr()** may be used to display the error text; **0** is returned on success.

pcap_freecode() is used to free up allocated memory pointed to by a *bpf_program* struct generated by **pcap_compile()** when that BPF program is no longer needed, for example after it has been made the filter program for a pcap structure by a call to **pcap_setfilter()**.

pcap_datalink() returns the link layer type; link layer types it can return include:

> **DLT_NULL** BSD LOOPBACK ENCAPSULATION; THE LINK LAYER HEADER IS A 4-BYTE FIELD, IN *HOST* BYTE ORDER, CONTAINING A PF_ VALUE FROM **SOCKET.H** FOR THE NETWORK-LAYER PROTOCOL OF THE PACKET.
>
> NOTE THAT ``HOST BYTE ORDER'' IS THE BYTE ORDER OF THE MACHINE ON WHICH THE PACKETS ARE CAPTURED, AND THE PF_ VALUES ARE FOR THE OS OF THE MACHINE ON WHICH THE PACKETS ARE CAPTURED; IF A LIVE CAPTURE IS BEING DONE, ``HOST BYTE ORDER'' IS THE BYTE ORDER OF THE MACHINE CAPTURING THE PACKETS, AND

THE PF_ VALUES ARE THOSE OF THE OS OF THE MACHINE CAPTURING THE PACKETS, BUT IF A ``SAVEFILE'' IS BEING READ, THE BYTE ORDER AND PF_ VALUES ARE *NOT* NECESSARILY THOSE OF THE MACHINE READING THE CAPTURE FILE.

pcap_list_datalinks() is used to get a list of the supported data link types of the interface associated with the pcap descriptor. **pcap_list_datalinks()** allocates an array to hold the list and sets **dlt_buf*. The caller is responsible for freeing the array. **-1** is returned on failure; otherwise, the number of data link types in the array is returned.

pcap_set_datalink() is used to set the current data link type of the pcap descriptor to the type specified by *dlt*. **-1** is returned on failure.

pcap_datalink_name_to_val() translates a data link type name, which is a **DLT_** name with the **DLT_** removed, to the corresponding data link type value. The translation is case-insensitive. **-1** is returned on failure.

pcap_datalink_val_to_name() translates a data link type value to the corresponding data link type name. NULL is returned on failure.

pcap_datalink_val_to_description() translates a data link type value to a short description of that data link type. NULL is returned on failure.

pcap_snapshot() returns the snapshot length specified when **pcap_open_live()** was called.

pcap_is_swapped() returns true if the current ``savefile'' uses a different byte order than the current system.

pcap_major_version() returns the major number of the file format of the savefile; **pcap_minor_version()** returns the minor number of the file format of the savefile. The version number is stored in the header of the savefile.

pcap_file() returns the standard I/O stream of the ``savefile,'' if a ``savefile'' was opened with **pcap_open_offline()**, or NULL, if a network device was opened with **pcap_open_live()**.

pcap_stats() returns 0 and fills in a **pcap_stat** struct. The values represent packet statistics from the start of the run to the time of the call. If there is an error or the underlying packet capture doesn't support packet statistics, -1 is returned and the error text can be obtained with **pcap_perror()** or **pcap_geterr()**. **pcap_stats()** is supported only on live captures, not on ``savefiles''; no statistics are stored in ``savefiles'', so no statistics are available when reading from a ``savefile''.

pcap_fileno() returns the file descriptor number from which captured packets are read, if a network device was opened with **pcap_open_live()**, or -1, if a ``savefile'' was opened with **pcap_open_offline()**.

pcap_perror() prints the text of the last pcap library error on **stderr**, prefixed by *prefix*.

pcap_geterr() returns the error text pertaining to the last pcap library error. **NOTE**: the pointer it returns will no longer point to a valid error message string after the **pcap_t** passed to it is closed; you must use or copy the string before closing the **pcap_t**.

pcap_strerror() is provided in case **strerror**(1) isn't available.

pcap_lib_version() returns a pointer to a string giving information about the version of the libpcap library being used; note that it contains more information than just a version number.

pcap_close() closes the files associated with *p* and deallocates resources.

pcap_dump_file() returns the standard I/O stream of the ``savefile'' opened by **pcap_dump_open().**

pcap_dump_flush() flushes the output buffer to the ``savefile,'' so that any packets written with **pcap_dump()** but not yet written to the ``savefile'' will be written. **-1** is returned on error, 0 on success.

pcap_dump_close() closes the ``savefile.''

ETHERNET STRUCTURE

```
          #include<net/ethernet.h>
           net/ethernet.h –defination for media access control protocol(mac)
#ifndef   _SYS_ETHERNET_H
#define  _SYS_ETHERNET_H
   #pragma ident"@(#)ethernet.h    1.21       05/06/08 SMI"

#ifdef     __cplusplus
extern "C" {
    #endif

    #define        ETHERADDRL    (6)                 /* ethernet address length in
octets */
    #define        ETHERFCSL     (4)                 /* ethernet FCS length in octets
*/

   /*
* Ethernet address - 6 octets
*/
   typedef uchar_t ether_addr_t[ETHERADDRL];

    /*
* Ethernet address - 6 octets
     */
    struct          ether_addr {
          ether_addr_t        ether_addr_octet;
};

/*
     * Structure of a 10Mb/s Ethernet header.
     */
struct    ether_header {
          struct   ether_addr          ether_dhost;
          struct   ether_addr          ether_shost;
          ushort_t            ether_type;
};

#define  ETHER_CFI         0

struct    ether_vlan_header {
          struct   ether_addr          ether_dhost;
struct    ether_addr          ether_shost;
          ushort_t            ether_tpid;
          ushort_t            ether_tci;
          ushort_t            ether_typ;};

#define  ETHERTYPE_PUP                      (0x0200) /* PUP protocol */
#define  ETHERTYPE_802_MIN        (0x0600) /* Min valid ethernet type */
                                   /* under IEEE 802.3 rules */
```

```
#define  ETHERTYPE_IP            (0x0800) /* IP protocol */
#define  ETHERTYPE_ARP                     (0x0806) /* Addr. resolution protocol */
 #define ETHERTYPE_REVARP        (0x8035) /* Reverse ARP */
 #define ETHERTYPE_AT            (0x809b) /* AppleTalk protocol */
 #define ETHERTYPE_AARP                    (0x80f3)  /* AppleTalk ARP */
 #define ETHERTYPE_IPV6                    (0x86dd) /* IPv6 */
 #define ETHERTYPE_SLOW                    (0x8809) /* Slow Protocol */
 #define ETHERTYPE_PPPOED        (0x8863) /* PPPoE Discovery Stage */
 #define ETHERTYPE_PPPOES        (0x8864) /* PPPoE Session Stage */
 #define ETHERTYPE_MAX                     (0xffff)    /* Max valid ethernet type */
 /*
 * The ETHERTYPE_NTRAILER packet types starting at ETHERTYPE_TRAIL have
  * (type-ETHERTYPE_TRAIL)*512 bytes of data followed
  * by an ETHER type (as given above) and then the (variable-length) header.
  */
 #define ETHERTYPE_TRAIL                   (0x1000) /* Trailer packet */
 #define ETHERTYPE_NTRAILER      (16)

 #define ETHERMTU                (1500)    /* max frame w/o header or fcs */
 #define ETHERMIN                (60)      /* min frame w/header w/o fcs */
 #define ETHERMAX                (1514)    /* max frame w/header w/o fcs */
 /*
 * Compare two Ethernet addresses - assumes that the two given
 * pointers can be referenced as shorts.  On architectures
 * where this is not the case, use bcmp instead.  Note that like
 * bcmp, we return zero if they are the SAME.
 */
 #if defined(__sparc) || defined(__i386) || defined(__amd64)
 #define ether_cmp(a, b) (((short *)b)[2] != ((short *)a)[2] || \
           ((short *)b)[1] != ((short *)a)[1] || \
           ((short *)b)[0] != ((short *)a)[0])
 #else
 #define ether_cmp(a, b) (bcmp((caddr_t)a, (caddr_t)b, 6))
 #endif

 /*
 * Copy Ethernet addresses from a to b - assumes that the two given
 * pointers can be referenced as shorts.  On architectures
 * where this is not the case, use bcopy instead.
  */
 #if defined(__sparc) || defined(__i386) || defined(__amd64)
 #define ether_copy(a, b) { ((short *)b)[0] = ((short *)a)[0]; \
           ((short *)b)[1] = ((short *)a)[1]; ((short *)b)[2] = ((short *)a)[2]; }
 #else
 #define ether_copy(a, b) (bcopy((caddr_t)a, (caddr_t)b, 6))
 #endif

 #ifdef    _KERNEL
 extern int localetheraddr(struct ether_addr *, struct ether_addr *);
 extern char *ether_sprintf(struct ether_addr *);
 extern int ether_aton(char *, uchar_t *);
 #else     /* _KERNEL */
 #ifdef  __STDC__
extern char *ether_ntoa(const struct ether_addr *);
extern struct ether_addr *ether_aton(const char *);
extern int ether_ntohost(char *, const struct ether_addr *);
extern int ether_hostton(const char *, struct ether_addr *);
extern int ether_line(const char *, struct ether_addr *, char *);
#else      /* __STDC__ */
extern char *ether_ntoa();
extern struct ether_addr *ether_aton();
extern int ether_ntohost();
extern int ether_hostton();
extern int ether_line();
#endif    /* __STDC__ */
#endif    /* _KERNEL */
#ifdef      __cplusplus
 }
#endif
#endif    /* _SYS_ETHERNET_H */
```

IP STRUCTURE

```
        #include <netinet/ip.h>
           netinet/ip.h - definitions for the Internet  Protocol (IP)
```

IP STRUCTURE DEFINATION:--

```
#ifndef _NETINET_IP_H_
#define <>_NETINET_IP_H_

/*
 * Definitions for internet protocol version 4.
 * Per RFC 791, September 1981.
 */
#define   <>IPVERSION      4

/*
 * Structure of an internet header, naked of options.
 */
struct <>ip {
#ifdef _IP_VHL
          u_char   ip_vhl;                   /* version << 4 | header length >> 2 */
#else
#if BYTE_ORDER == LITTLE_ENDIAN
          u_int    ip_hl:4,          /* header length */
                   ip_v:4;                   /* version */
#endif
#if BYTE_ORDER == BIG_ENDIAN
          u_int    ip_v:4,                   /* version */
                   ip_hl:4;          /* header length */
#endif
#endif /* not _IP_VHL */
          u_char   ip_tos;                   /* type of service */
          u_short  ip_len;                   /* total length */
          u_short  ip_id;                    /* identification */
          u_short  ip_off;                   /* fragment offset field */
#define   <>IP_RF 0x8000                     /* reserved fragment flag */
#define   <>IP_DF 0x4000                     /* dont fragment flag */
#define   <>IP_MF 0x2000                     /* more fragments flag */
#define   <>IP_OFFMASK 0x1fff                /* mask for fragmenting bits */
          u_char   ip_ttl;                   /* time to live */
          u_char   ip_p;                     /* protocol */
          u_short  ip_sum;                   /* checksum */
          struct   in_addr ip_src,ip_dst;    /* source and dest address */
};

#ifdef _IP_VHL
```

```
#define   <>IP_MAKE_VHL(v, hl)      ((v) << 4 | (hl))
#define   <>IP_VHL_HL(vhl)                    ((vhl) & 0x0f)
#define   <>IP_VHL_V(vhl)           ((vhl) >> 4)
#define   <>IP_VHL_BORING                     0x45
#endif

#define   <>IP_MAXPACKET            65535              /* maximum packet size */

/*
 * Definitions for IP type of service (ip_tos)
 */
#define   <>IPTOS_LOWDELAY                    0x10
#define   <>IPTOS_THROUGHPUT        0x08
#define   <>IPTOS_RELIABILITY       0x04
#define   <>IPTOS_MINCOST                     0x02
/* ECN bits proposed by Sally Floyd */
#define   <>IPTOS_CE                0x01      /* congestion experienced */
#define   <>IPTOS_ECT               0x02      /* ECN-capable transport */


/*
 * Definitions for IP precedence (also in ip_tos) (hopefully unused)
 */
#define   <>IPTOS_PREC_NETCONTROL                      0xe0
#define   <>IPTOS_PREC_INTERNETCONTROL                 0xc0
#define   <>IPTOS_PREC_CRITIC_ECP                      0xa0
#define   <>IPTOS_PREC_FLASHOVERRIDE0x80
#define   <>IPTOS_PREC_FLASH                  0x60
#define   <>IPTOS_PREC_IMMEDIATE                       0x40
#define   <>IPTOS_PREC_PRIORITY                        0x20
#define   <>IPTOS_PREC_ROUTINE                         0x00

/*
 * Definitions for options.
 */
#define   <>IPOPT_COPIED(o)                   ((o)&0x80)
#define   <>IPOPT_CLASS(o)                    ((o)&0x60)
#define   <>IPOPT_NUMBER(o)                   ((o)&0x1f)

#define   <>IPOPT_CONTROL                     0x00
#define   <>IPOPT_RESERVED1                   0x20
#define   <>IPOPT_DEBMEAS                     0x40
#define   <>IPOPT_RESERVED2                   0x60

#define   <>IPOPT_EOL               0                  /* end of option list */
#define   <>IPOPT_NOP               1                  /* no operation */

#define   <>IPOPT_RR                7                  /* record packet route */
#define   <>IPOPT_TS                68                 /* timestamp */
#define   <>IPOPT_SECURITY                    130                /* provide s,c,h,tcc */
#define   <>IPOPT_LSRR              131                /* loose source route */
```

```
#define   <>IPOPT_SATID              136                  /* satnet id */
#define   <>IPOPT_SSRR               137                  /* strict source route */
#define   <>IPOPT_RA                 148                  /* router alert */

/*
 * Offsets to fields in options other than EOL and NOP.
 */
#define   <>IPOPT_OPTVAL                      0                    /* option ID */
#define   <>IPOPT_OLEN               1                    /* option length */
#define <>IPOPT_OFFSET               2                    /* offset within option */
#define   <>IPOPT_MINOFF                      4                    /*  min  value  of  above
*/

/*
 * Time stamp option structure.
 */
struct      <>ip_timestamp {
            u_char   ipt_code;             /* IPOPT_TS */
            u_char   ipt_len;              /* size of structure (variable) */
            u_char   ipt_ptr;              /* index of current entry */
#if BYTE_ORDER == LITTLE_ENDIAN
            u_int    ipt_flg:4,            /* flags, see below */
                     ipt_oflw:4;                    /* overflow counter */
#endif
#if BYTE_ORDER == BIG_ENDIAN
            u_int    ipt_oflw:4,                    /* overflow counter */
                     ipt_flg:4;            /* flags, see below */
#endif
            union <>ipt_timestamp {
                     n_long   ipt_time[1];
                     struct   <>ipt_ta {
                              struct in_addr ipt_addr;
                              n_long ipt_time;
                     } ipt_ta[1];
            } ipt_timestamp;
};

/* flag bits for ipt_flg */
#define   <>IPOPT_TS_TSONLY                   0                    /* timestamps only */
#define   <>IPOPT_TS_TSANDADDR                1                    /*     timestamps    and
addresses */
#define   <>IPOPT_TS_PRESPEC      3                    /* specified modules only */

/* bits for security (not byte swapped) */
#define   <>IPOPT_SECUR_UNCLASS          0x0000
#define   <>IPOPT_SECUR_CONFID           0xf135
#define   <>IPOPT_SECUR_EFTO    0x789a
#define   <>IPOPT_SECUR_MMMM 0xbc4d
#define   <>IPOPT_SECUR_RESTR 0xaf13
#define   <>IPOPT_SECUR_SECRET           0xd788
#define   <>IPOPT_SECUR_TOPSECRET        0x6bc5
```

```
/*
 * Internet implementation parameters.
 */
#define   <>MAXTTL                  255               /*    maximum    time    to    live
(seconds) */
#define   <>IPDEFTTL       64                  /* default ttl, from RFC 1340 */
#define   <>IPFRAGTTL      60                  /* time to live for frags, slowhz */
#define   <>IPTTLDEC       1                   /* subtracted when forwarding */

#define   <>IP_MSS                  576               /*   default   maximum   segment
size */

#endif
```

TCP STRUCTURE

#include <netinet/tcp.h>

NETINET/TCP.H - DEFINITIONS FOR THE INTERNET TRANSMISSION CONTROL PROTOCOL (TCP)

THE <*NETINET/TCP.H*> HEADER SHALL DEFINE THE FOLLOWING MACRO FOR USE AS A SOCKET OPTION AT THE IPPROTO_TCP LEVEL:

TCP_NODELAY
AVOID COALESCING OF SMALL SEGMENTS.

The macro shall be defined in the header. The implementation need not allow the value of the option to be set via retrieved via

TCP STRUCTURE DEFINATION:--

```
#ifndef _NETINET_TCP_H_
#define <>_NETINET_TCP_H_

typedef   u_int32_t <>tcp_seq;
typedef u_int32_t <>tcp_cc;                          /* connection count per rfc1644 */

#define <>tcp6_seq          tcp_seq  /* for KAME src sync over BSD*'s */
#define <>tcp6hdr                    tcphdr     /* for KAME src sync over BSD*'s */

/*
 * TCP header.
 * Per RFC 793, September, 1981.
 */
struct <>tcphdr {
            u_short   th_sport;             /* source port */
            u_short   th_dport;             /* destination port */
            tcp_seq   th_seq;                          /* sequence number */
            tcp_seq   th_ack;                          /* acknowledgement number */
#if BYTE_ORDER == LITTLE_ENDIAN
            u_int       th_x2:4,              /* (unused) */
                        th_off:4;             /* data offset */
#endif
#if BYTE_ORDER == BIG_ENDIAN
            u_int       th_off:4,             /* data offset */
                        th_x2:4;              /* (unused) */
#endif
            u_char    th_flags;
#define   <>TH_FIN          0x01
#define   <>TH_SYN          0x02
#define   <>TH_RST          0x04
#define   <>TH_PUSH         0x08
#define   <>TH_ACK          0x10
#define   <>TH_URG          0x20
#define   <>TH_ECE          0x40
#define   <>TH_CWR          0x80
#define   <>TH_FLAGS
            (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)

            u_short   th_win;                          /* window */
            u_short   th_sum;                          /* checksum */
            u_short   th_urp;                          /* urgent pointer */
};

#define   <>TCPOPT_EOL               0
#define   <>TCPOPT_NOP               1
#define   <>TCPOPT_MAXSEG                      2
```

```
#define   <>TCPOLEN_MAXSEG              4
#define <>TCPOPT_WINDOW                 3
#define   <>TCPOLEN_WINDOW              3
#define <>TCPOPT_SACK_PERMITTED         4                   /* Experimental */
#define   <>TCPOLEN_SACK_PERMITTED  2
#define <>TCPOPT_SACK           5                 /* Experimental */
#define <>TCPOPT_TIMESTAMP    8
#define   <>TCPOLEN_TIMESTAMP                     10
#define   <>TCPOLEN_TSTAMP_APPA                   (TCPOLEN_TIMESTAMP+2)  /*
appendix A */
#define   <>TCPOPT_TSTAMP_HDR                     \

(TCPOPT_NOP<<24|TCPOPT_NOP<<16|TCPOPT_TIMESTAMP<<8|TCPOLEN_TIME
STAMP)

#define   <>TCPOPT_CC           11                /* CC options: RFC-1644 */
#define <>TCPOPT_CCNEW                  12
#define <>TCPOPT_CCECHO                 13
#define     <>TCPOLEN_CC                          6
#define     <>TCPOLEN_CC_APPA           (TCPOLEN_CC+2)
#define     <>TCPOPT_CC_HDR(ccopt)                \
   (TCPOPT_NOP<<24|TCPOPT_NOP<<16|(ccopt)<<8|TCPOLEN_CC)

/*
 * Default maximum segment size for TCP.
 * With an IP MSS of 576, this is 536,
 * but 512 is probably more convenient.
 * This should be defined as MIN(512, IP_MSS - sizeof (struct tcpiphdr)).
 */
#define   <>TCP_MSS       512

/*
 * Default maximum segment size for TCP6.
 * With an IP6 MSS of 1280, this is 1220,
 * but 1024 is probably more convenient. (xxx kazu in doubt)
 * This should be defined as MIN(1024, IP6_MSS - sizeof (struct tcpip6hdr))
 */
#define   <>TCP6_MSS      1024

#define   <>TCP_MAXWIN  65535     /* largest value for (unscaled) window */
#define   <>TTCP_CLIENT_SND_WND         4096      /* dflt send window for T/TCP
client */

#define <>TCP_MAX_WINSHIFT      14          /* maximum window shift */

#define <>TCP_MAXBURST                  4           /* maximum segments in a burst
*/

#define <>TCP_MAXHLEN (0xf<<2)  /* max length of header in bytes */
#define <>TCP_MAXOLEN (TCP_MAXHLEN - sizeof(struct tcphdr))
                                        /* max space left for options */
```

```
/*
 * User-settable options (used with setsockopt).
 */
#define   <>TCP_NODELAY            0x01       /* don't delay send to coalesce packets */
#define   <>TCP_MAXSEG 0x02       /* set maximum segment size */
#define <>TCP_NOPUSH   0x04       /* don't push last block of write */
#define <>TCP_NOOPT     0x08       /* don't use TCP options */

#endif
```

REFERECNES

http://www.skullbox.net/firewalls.php
http://support.microsoft.com/?kbid=321050
http://www.vicomsoft.com/knowledge/reference/firewalls1.html
http://www.firewall-software.com/firewall_faqs/types_of_firewall.html
http://www.tldp.org/HOWTO/html_single/IP-Masquerade-HOWTO/#IPMASQ-INTRO1.0
http://www.tldp.org/
http://www.netfilter.org/projects/iptables/index.html
http://www.securityfocus.com/infocus/1674
http://www.google.co.in/search?hl=en&lr=&defl=en&q=define:Spoofing&sa=X&oi=glossary_definition&ct=title
http://www.iss.net/security_center/advice/Underground/Hacking/Methods/Technical/Spoofing/default.htm
http://www.google.co.in/search?hl=en&lr=&defl=en&q=define:MAC+Address&sa=X&oi=glossary_definition&ct=title
http://compnetworking.about.com/od/networkprotocolsip/l/aa062202a.htm
http://www.artsci.wustl.edu/ASCC/documentation/macaddrss.html
http://en.wikipedia.org/wiki/IP_address
http://www.google.co.in/search?hl=en&lr=&defl=en&q=define:IP+Address&sa=X&oi=glossary_definition&ct=title
http://www.google.co.in/search?hl=en&lr=&defl=en&q=define:Private+IP+address&sa=X&oi=glossary_definition&ct=title
http://www.duxcw.com/faq/network/privip.htm
http://en.wikipedia.org/wiki/Private_IP_address
http://www.auditmypc.com/internal-ip.html
http://kb.iu.edu/data/aijr.html
http://www.xo.com/products/smallgrowing/internet/dsl/glossary/publicip.html
http://www.vicomsoft.com/glossary/addresses.html