::::TCP\IP: A Mammoth Description By Ankit Fadia ankit@bol.net.in::::

TCP\IP or Transmission Control Protocol \ Internet Protocol is a stack or collection of various protocols. A protocol is basically the commands or instructions using which two computers within a local network or the Internet can exchange data or information and resources.

Transmission Control Protocol \ Internet Protocol or the TCP\IP was developed around the time of the ARPAnet. It is also known as the Protocol Suite. It consists of various protocols but as the TCP (Transmission Control Protocol) and the IP (Internet Protocol) are the most, well known of the suite of protocols, the entire family or suite is called the TCP\IP suite.

The TCP\ IP Suite is a stacked suite with various layers stacked on each other, each layer looking after one aspect of the data transfer. Data is transferred from one layer to the other. The Entire TCP\ IP suite can be broken down into the below layers-:

Layer Name	Protocol
Link Layer (Hardware, Ethernet)	ARP, RARP, PPP, Ether
Network Layer(The Invisible Layer)	IP, ICMP
Transport Layer	UDP, TCP
Application Layer(The Visible Layer)	The Actual running Applications like-: FTP client, Browser
Physical Layer (Not part of TCP \IP)	Physical Data Cables, Telephone wires

Data travels from the Link Layer down to the Physical Layer at the source and at the destination it travels from the Physical Layer to the Link Layer. We will later discuss what each layer and each protocol does.

The TCP\IP suite not only helps to transfer data but also has to correct various problems that might occur during the data transfer. There are basically two types of most common errors that might occur during the process of data transfer. They are-:

Data Corruption -: In this kind of error, the data reaches the destination after getting corrupted.
Data Loss -: In this kind of error, the entire collection of packets which constitute the data to be transferred
does not reach the destination.

TCP\IP expects such errors to take place and has certain features which prevent, such error which might occur.

Checksums-: A checksum is a value (Normally, a 16 Bit Value) that is formed by summing up the Binary Data in the used program for a given data block. The program being used is responsible for the calculation of the Checksum value. The data being sent by the program sends this calculated checksum value, along with the data packets to the destination. When the program running at the destination receives the data packets, it re-calculates the Checksum value. If the Checksum value calculated by the Destination program matches with the Checksum Value attached to the Data Packets by the Source Program match, then the data transfer is said to be valid and error free. Checksum is calculated by adding up all the octets in a datagram.

Packet Sequencing-: All data being transferred on the net is broken down into packets at the source and joined together at the destination. The data is broken down into packets in a particular sequence at the source. This means that, for example, the first byte has the first sequence number and the second byte the second sequence number and so on. These packets are free to travel independently on the net, so sometimes, when the data packets reach the destination they arrive, out of sequence, which means that the packet which had the first sequence number attached to it does not reach the destination first. Sequencing defines the order in which the hosts receive the data packets or messages. The application or the layer running at the destination automatically builds up the data from the sequence number in each packet.
The source system breaks the data to be transferred into smaller packets and assigns each packet a unique sequence number. When the destination gets the packets, it's starts rearranging the packets by reading the sequence numbers of each packet to make the data received usable.

For example, say you want to transfer a 18000 octet file. Not all networks can handle the entire 18000 octet packets at a time. So the huge file is broken down into smaller say 300 octet packets. Each packet has been assigned a unique sequence number. Now when the packets reach the destination the packets are put back together to get the usable data. Now during the transportation process, as the packets can move independently on the net, it is possible that the packet 5 will arrive at the destination before packet 4 arrives. In such a situation, the sequence numbers are used by the destination to rearrange the data packets in such a way that even if Data packet 5 arrived earlier, Packet 4 will always precede Packet 5.

A data can easily be corrupted while it is being transferred from the source to the destination. Now if a error control service is running then if it detects data corruption, then it asks the source to re-send the packets of data. Thus only non corrupted data reaches the destination. An error control service detects and controls the same two types of errors-:

1.) Data Loss
2.) Data Corruption

The Checksum values are used to detect if the data has been modified or corrupted during the transfer from source to destination or any corruption in the communication channel which may have caused data loss. Data Corruption is detected by the Checksum Values and by performing Cyclic Redundancy Checks (CRC 's). CRC 's too like the Checksums are integer values but require intensely advanced calculation and hence are rarely used.

There is yet another way of detecting data corruption-: Handshaking.

This feature ensures demands that both the source and destination must transmit and receive acknowledgement messages, that confirm transfer of uncorrupted data. Such acknowledgement messages are known as ACK messages.

Let's take an example of a typical scenario of data transfer between two systems.
Source Sends MSG1 to Destination. It will not send MSG2 to Destination unless and until it gets the MSG ACK and destination will not send more requests for data or the next request message (MSG2) unless it gets the ACK from Source confirming that the MSG1 ACK was received by it. If the source does not get a ACK message from the destination, then something which is called a timed-out occurs and the source will re send the data to destination.

So this means that if A sends a data packet to B and B checksums the data packet and finds the data corrupted, then it can simply delete for a time out to take place. Once the time out takes place, A will re send the data packet to B. But this kind of system of deleting corrupt data is not used as it is inefficient and
time consuming.

Instead of deleting the corrupt data and waiting for a time out to take place, the destination (B) sends a not acknowledged or NACK message to source(A). When A gets the NACK message, instead of waiting for a time out to take place, it straightaway resends the data packet.

An ACK message of 1000 would mean that all data up to 1000 octets has been received till now.

TCP/ IP is a layered suite of protocols. All layers are equally important and with the absence of even a single layer, data transfer would not have been possible. Each TCP/ IP layer contributes to the entire process of data transfer. An excellent example, is when you send an email. For sending mail there is a separate protocol, the SMTP protocol which belongs to the Application layer. The SMTP Application protocol like all other application layer protocols assumes that there is a reliable connection existing between the two computers. For the SMTP application protocol to do what it is designed for, i.e. to send mail, it requires the existence of all other Layers as well. The Physical Layer i.e. cables and wires is required to transport the data physically. The Transmission Control Protocol or the TCP protocol which belongs to the Transport Layer is needed to keep track of the number of packets sent and for error correction. It is this protocol that makes sure that the data reaches the other end. The TCP protocol is called
by the Application Protocol to ensure error free communication between the source and destination. For the TCP layer to do its work properly i.e. to ensure that the data packets reach the destination, it requires the existence of the Internet Protocol or IP. The IP protocol contains the Checksum and Source and Destination IP address.

You may wonder why do we need different protocols like TCP and IP and why not bundle them into the same Application protocol.? The TCP protocol contains commands or functions which are needed by various application protocols like FTP, SMTP and also HTTP. The TCP protocol also calls on the IP protocol, which in turn contains commands or functions which some application protocols require while others don?t. So rather than bundling the entire TCP and IP protocol set into specific application protocols, it is better to have different protocols which are called whenever required.

The Link Layer which is the Hardware or Ethernet layer is also needed for transportation of the data packets. The PPP or the Point to Point Protocol belongs to this layer. Before we go on let's get accustomed with certain TCP\IP terms. Most people get confused between datagrams and packets and think that they are one and the same thing . You see, a datagram is a unit of data which is used by various protocols and a packet is a physical object or thing which moves on a physical medium like a wire. There is a remarkable difference between a Packet and a Datagram, but it is beyond the scope of this book. To make things easier I will use only the term datagram (Actually this is the official term.)while discussing various protocols.

Two different main protocols are involved in transporting packets from source to destination.

1.) The Transmission Control Protocol or the TCP Protocol
2.) The Internet Protocol or the IP protocol.

Besides these two main protocols, the Physical Layer and the Ethernet Layer are also indispensable to data transfer.

THE TRANSPORT LAYER

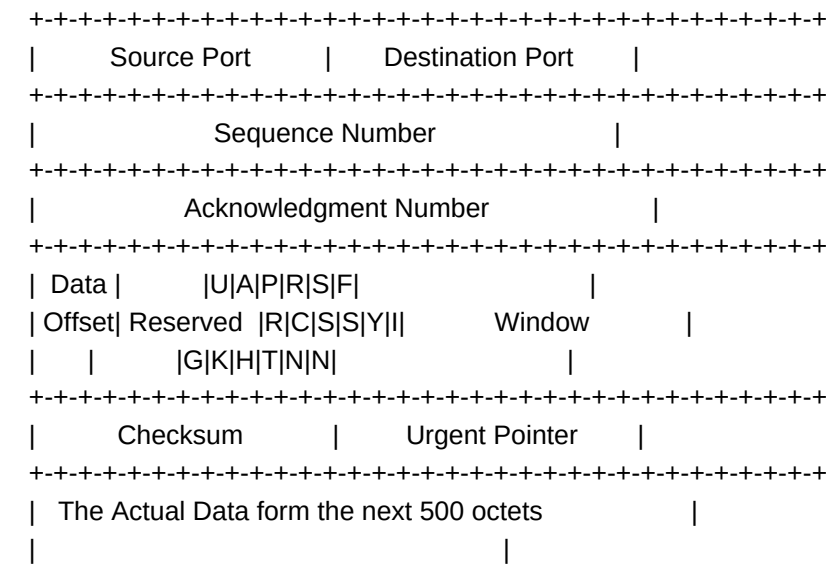
The TCP protocol

The Transmission Control Protocol is responsible for breaking up the data into smaller datagrams and putting the datagrams back to form usable data at the destination. It also resends the lost datagrams to destination where the received datagrams are reassembled in the right order. The TCP protocol does the bulk of work but without the IP protocol, it cannot transfer data.

Let's take an example to make things more clearer. Let's say your Internet Protocol Address or IP address is xxx.xxx.xxx.xxx or simply x and the destination's IP is yyy.yyy.yyy.yyy or simply y. Now As soon as the three-way connection is established between x and y, x knows the destination IP address and also the Port to which it is connected to. Both x and y are in different networks which can handle different sized packets. So in order to send datagrams which are in receivable size, x must know what is the maximum datagram size which y can handle. This too is determined by both x and y during connection time.

So once x knows the maximum size of the datagram which y can handle, it breaks down the data into smaller chunks or datagrams. Each datagram has it's own TCP header which too is put by TCP.
A TCP Header contains a lot of information, but the most important of it is the Source and Destination IP and Port numbers and yes also the sequence number.

****
HACKING TRUTH: Learn more about Ports, IP's, Sockets in the Net Tools Manual
****
The source which is your computer(x) now knows what the IP Addresses and Port Numbers of the Destination and Source computers are. It now calculates the Checksum value by adding up all the octets of the datagram and puts the final checksum value to the TCP Header. The different octets and not the datagrams are then numbered. An octet would be a smaller broken down form of the entire data. TCP then puts all this information into the TCP header of each datagram. A TCP Header of a datagram would finally look like -:



There are certain new fields in the TCP header which you may not know off. Let's see what these new fields signify. The Windows field specifies the octets of new data which is ready to be processed. You see not all computers connected to the Internet run at the same speed and to ensure that a faster system does not send datagrams to a slow system at a rate which is faster than it can handle, we use the Window field. As the computer receives data , the space in the Window field gets decreased indicating that the receiver has received the data. When it reaches zero the sender stops sending further packets. Once the receiver finishes processing the received data, it increases the Window field, which in turn indicates that the receiver has processed the earlier sent data and is ready to receive more chunks of data.

The Urgent Field tells the remote computer to stop processing the last octet and instead receive the new octet. This is normally not commonly used.

The TCP protocol is a reliable protocol, which means that we have a guarantee that the data will arrive at

the destination properly and without any errors. It ensures that the data being received by the receiving end is arranged in the same correct order in which it was sent.

The TCP Protocol relies on a virtual circuit between the client and the host. The circuit is opened via a 3 part process known as the three part handshake. It supports full duplex transportation of data which means that it provides a path for two way data transfer. Hence using the TCP protocol, a computer can send and receive datagrams at the same time.

Some common flags of TCP are-:

RST [RESET]- Resets the connection.
PSH [PUSH] - Tells receiver to pass all queued data to the application running.
FIN [FINISH] - Closes connection following the 4 step process.
SYN Flag - means that the machine sending this flag wants to establish a three way handshake i.e. a TCP connection. The receiver of a SYN flag usually responds with an ACK message.

So now we are in a position to represent a three way TCP Handshake:

A <---SYN---> B
A <---SYN/ACK? B
A <---ACK---> B

A sends a SYN flag to B saying " I want to establish a TCP connection", B responds to the SYN with the ACK to the SYN flag. A again responds to the ACK sent by B with another ACK.

Read RFC 793 for further in depth details about the TCP protocol.

The User Datagram Protocol or the UDP Protocol

The User Data protocol or the UDP is yet another protocol which is a member of the Transport Layer. TCP is the standard protocol used by all systems for communications. TCP is used to break down the data to be transported into smaller datagrams, before they (the datagrams) are sent across a network. Thus we can say that TCP is used where more than a single or multiple datagrams are involved.

Sometimes, the data to be transported is able to fit into a single datagram. We do not need to break the data into smaller datagrams as the size of the data is pretty small. The perfect example of such data is the DNS system. To send out the query for a particular domain name, a single datagram is more than enough. Also the IP that is returned by the Domain Name Server does not require more than one datagram for transportation. So in such cases instead of making use of the complex TCP protocol, applications fall back to the UDP protocol.

The UDP protocol works almost the way TCP works. But the only differences being that TCP breaks the data to be transferred into smaller chunks, does sequencing by inserting a sequence number in the header and no error control. Thus we can conclude by saying that the UDP protocol is an unreliable protocol with no way to confirm that the data has reached the destination.

The UDP protocol does insert a USP header to the single datagram it is transporting. The UDP header contains the Source and Destination IP Addresses and Port Numbers and also the Checksum value. The UDP header is comparatively smaller than the TCP Header.

It is used by those applications where small chunks of data are involved. It offers services to the User's Network Applications like NFS(Network File Sharing) and SNMP.

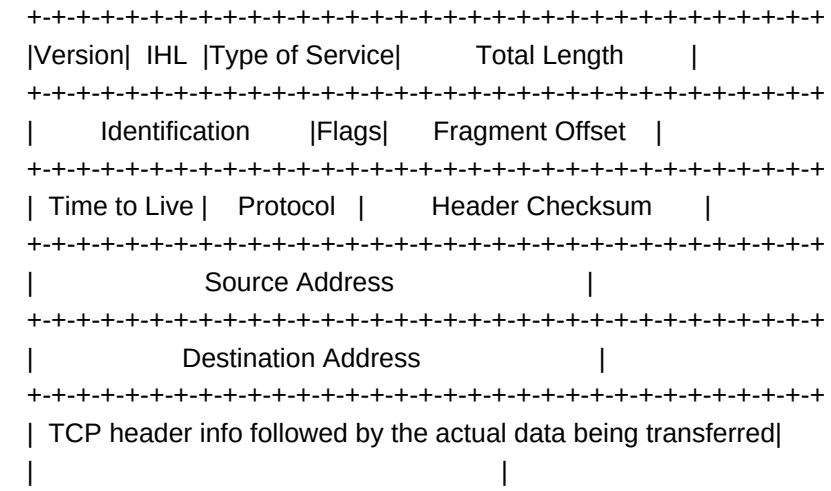
Read RFC 768 for further in depth details about the UDP protocol.

THE NETWORK LAYER

The IP Protocol

Both the TCP and the UDP protocols, after inserting the headers to the datagram(s) given to them pass them to the Internet Protocol or the IP Protocol. The main job of the IP protocol is to find a way of transporting the datagrams to the destination receiver. It does not do any kind of error checking.

The IP protocol too adds it's own IP Header to each datagram. The IP header contains the source and destination IP addresses, the protocol number and yet another checksum. The IP header of a particular datagram looks like-:



The Source and destination IP addresses and needed so that?well it is obvious isn't it? The Protocol number is added so that the IP protocol knows to which Transport Protocol the datagram has to be passed. You see various Transport Protocols are used like for example TCP or UDP. So this protocol number is inserted to tell IP the protocol to which the datagram has to be passed.

It too inserts it's own Checksum value which is different from the Checksum Value inserted by the Transport Protocols. This Checksum has to be inserted as without it the Internet Protocol will not be able to verify if the Header has been damaged in the transfer process and hence the datagram might reach a wrong destination. The Time to Live field specifies a value which is decreased each time the datagram passes through a network. Remember Tracert?

The Internet Protocol Header contains other fields as well, but they are quite advanced and cannot be included in a manual which gives an introduction to the TCP\IP protocol. To learn more about the IP protocol read RFC 791.

The Internet Control Message Protocol or the ICMP

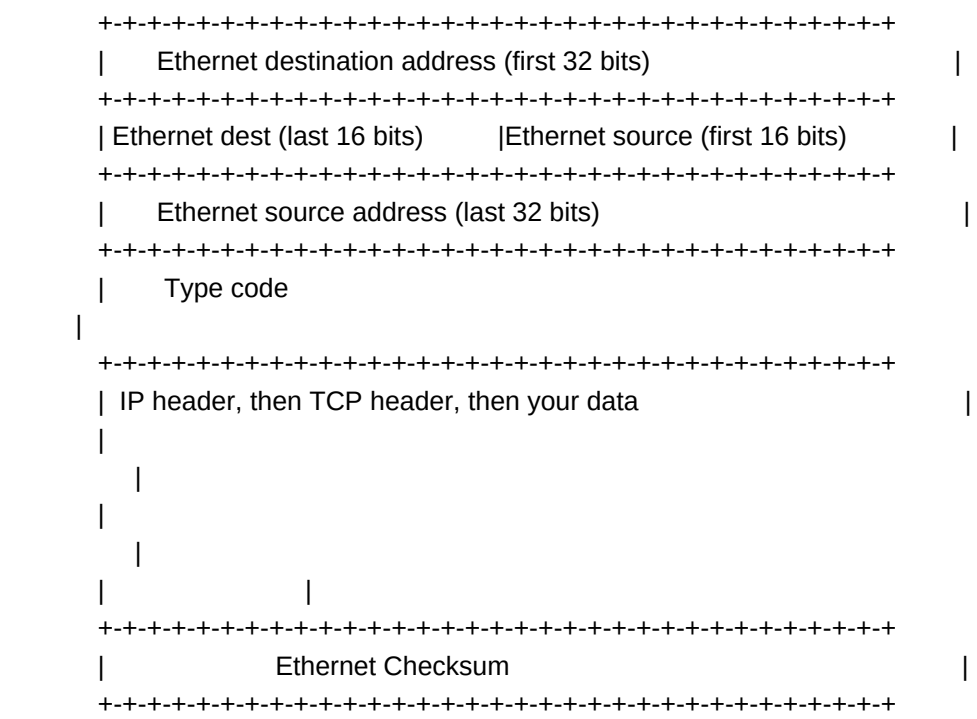
The ICMP protocol allows hosts to transfer information on errors that might have occurred during the data transfer between two hosts. It is basically used to display error messages about errors that might occur during the data transfer. The ICMP is a very simple protocol without any headers. It is most commonly used to diagnose Network Problems. The famous utility PING is a part of the ICMP protocol. ICMP requests do not require the user or application to mention any port number as all ICMP requests are answered by the Network Software itself. The ICMP protocol too handles only a single datagram. That's why we say in PING only a single datagram is sent to the remote computer. This protocol can remote many network problems like Host Down, Congested Network etc

Read RFC 792 for further in depth details about the ICMP protocol.

The Link Layer

Almost all networks use Ethernet. Each machine in a network has it's own IP address and it's Ether Address. The Ether Address of a computer is different than it's IP address. An Ether Address is a 42 bit address while the IP address is only a 32 bit address. A Network must know which computer to deliver the datagram to. Right? For this the Ether Header is used.

The Ether Header is a 14 octet header that contains the Source and Destination Ethernet address, and a type code. Ether too calculates it's own Checksum value. The Type code relates to the protocol families to be used within the Network. The Ether Layer passes the datagram to the protocol specified by this field after inserting the Ether Header. There is simply no connection between the Ethernet Address and the IP address of a machine. Each machine needs to have a Ethernet to IP address translation table on its hard disk.



Address Resolution Protocol or ARP

Data before being transmitted across the Internet or across a local network is broken down into smaller Packets which are suitable for transfer over the net. These packets have the Source and Destination IP's but for the transfer to take place the suitable Hardware Addresses or the MAC addresses must also be known. That is where ARP comes in.

To get the Hardware MAC addresses, ARP or Address Resolution Protocol sends a request message. The Router replies with the Hardware Address. It is similar to the DNS and it too has a cache. This cache can be a bit vulnerable as a Hacker could forge a connection from a remote machine claiming to be one of the cached locations. So we can conclude that ARP translates IP's into Ethernet Addresses. One thing to remember about ARP is that it only translates outgoing packets.

There is also something called the RARP which is an abbreviation for Reverse Address Resolution Protocol, which like the name says does exactly reverse of what ARP does.

There is simply no algorithm to get the Ethernet Address from the IP Address. To carry out such translations, each computer has a file which has a table with rows for each computer and two columns for their corresponding IP address and Ethernet Address. The File is somewhat like the following-:

	Internet Protocol Address	Ethernet Address
Computer Name	xxx.xy.yy.yx	08-00-39-00-2F-C3

Say there are a system in a Network (A) and an unidentified system (B) contacts it. Now A only knows the IP address of B. Now A will first try to identify whether B is the same network so that it can directly communicate via Ethernet. So it will first check the IP to MAC address translation table which it has. If it finds the IP in the table then well and good and A will establish a connection with B via Ethernet.

On the Other hand if A does not find any match for the specific IP, it will send out a request in the form of a 'Broadcast'. All computers within the Network will receive this broadcast and will search their own IP to MAC translation table and will reply with the necessary MAC address. A basic difference between an Ip address and MAC address is that an IP is the form xxx.xxx.xxx.xxx and a MAC address is in the form xx:xx:xx:xx:xx:xx and one is 32 bit while the other is 40 bit.

Read RFC 826 for further in depth details about the ARP protocol.

Application Layer

Till now you have learnt how data is broken down into smaller chunks, and transferred to the destination, where the chunks are rearranged. But there is yet another aspect to a successful data transfer process, which we have not discussed yet: The Application Protocols and the Application Layer itself. A host which receives datagrams has many applications or services (daemons) running which are ready to establish a TCP connection and accept a message. Datagrams travelling on the Internet must know which application they have to establish connection with, which application they have to send the message to. A typical web server will have the FTP daemon, the HTTP daemon, the POP daemon, and the SMTP daemon running. Wouldn't the datagrams get confused as to which daemon to send the message to.

For the datagrams to know which computer to send the message to, we have IP addresses. The datagram knows what daemon or application to send the message to by the Port Number attached to the IP address of the Destination. A TCP address is actually fully described by 4 numbers; The IP address of the Source and Destination and the TCP Port Numbers of each end to which data is to be sent. These numbers are found in the TCP Header.

To make it simpler to understand I have included an excerpt from the Net Tools Chapter:

What is all the hype about socket programming? What exactly are sockets?

TCP\IP or Transmission Control Protocol\ Internet Protocol is the language or the protocol used by computers to communicate with each other over the Internet. Say a computer whose IP address is 99.99.99.99 wants to communicate with another machine whose IP address is 98.98.98.98 then would will happen?

The machine whose IP is 99.99.99.99 sends a packet addressed to another machine whose IP is 98.98.98.98. When 98.98.98.98 receives the packet then it verifies that it got the message by sending a signal back to 99.99.99.99.But say the person who is using 99.99.99.99 wants to have simultaneously more than one connections to 98.98.98.98.....then what will happen? Say 99.99.99.99 wants to connect to the FTP daemon and download a file by FTP and at the same time it wants to connect to 98.98.98.98's website i.e. The HTTP daemon. Then 98.98.98.98. will have 2 connects with 99.99.99.99 simultaneously. Now how can 98.98.98.98.distinguish between the two connections...how does 98.98.98.98. know which is for the FTP daemon and which for the HTTP daemon? If there was no way to distinguish between the two connections then they would both get mixed up and there would be a lot of chaos with the message meant for the HTTP daemon going to the FTP daemon. To avoid such confusion we have ports. At each port a particular service or daemon is running by default. So now that the 99.99.99.99 computers knows

which port to connect to, to download a FTP file and which port to connect to, to download the web page, it will communicate with the 98.98.98.98 machine using what is known as the socket pair which is a combination of an IP address and a Port. So in the above case the message which is meant for the FTP daemon will be addressed to 98.98.98.98 : 21 (Notice the colon and the default FTP port suceeding it.). So that the receiving machine i.e. 98.98.98.98 will know for which service this message is meant for and to which port it should be directed to.

In TCP\IP or over the Internet all communication is done using the Socket pair i.e. the combination of the IP address and the port.

****
HACKING TRUTH: Learn More about Ports, IP addresses and Sockets by reading the Net Tools Chapter.
****
The Application Layers basically consists of the Applications running on your computer and the Applications running on the host to which you are connected. Say you are viewing the Hotmail Site, then the application layer comprises of the Web Browser running on your computer and the HTTP daemon running at Hotmail's server and the Application Protocol being used to communicate is HyperText Transfer Protocol.

As soon as a TCP connection is established the Applications running on Each end decide the language or protocol to be used to communicate and send datagrams.

IP Spoofing Torn Apart

IP spoofing is the most exciting topic you will hear wannabe hackers talking about. It is also a subject about which no one knows much. Before we continue I would like to tell you that IP Spoofing is quite difficult to understand and a lot of people have trouble understanding how it is done. The other downside it has is the fact that it can almost not be done using a Windows system and a system administrator can easily protect his system from IP spoofing

So what is IP Spoofing? IP Spoofing is a trick played on servers to fool the target computer into thinking that it is receiving data from a source other than you. This in turn basically means to send data to a remote host so that it believes that the data is coming from a computer whose IP address is something other than yours. Let's take an example to make it clear:

Your IP is : 203.45.98.01 (REAL)
IP of Victim computer is: 202.14.12.1 (VICTIM)
IP you want data to be sent from: 173.23.45.89 (FAKE)

Normally sitting on the computer whose IP is REAL, the datagrams you send to VICTIM will appear to have come from REAL. Now consider a situation in which you want to send a datagram to VICTIM and make him believe that it came from a computer whose IP is FAKE. This is when you perform IP Spoofing.

The Main problem with IP Spoofing is that even if you are able to send a spoofed datagram to the remote host, the remote host will reply not to your real IP but to the Fake IP you made your datagram seem to have come from. Getting confused? Read the following example to clear up your mind.

Taking the same IP's as in the last example, consider the following scenario. Now, if REAL connects to VICTIM, after the standard three way handshake has taken place, and VICTIM sends an ACK message to REAL. Now if you spoof you IP, to say FAKE, then VICTIM will try to establish a TCP connection and will send an ACK message to FAKE. Now lets assume that FAKE is alive, then as it had not requested the ACK message (sent by VICTIM to FAKE) it will reply with a NACK message which would basically end the connection and no further communication between FAKE and VICTIM would take place. Now if FAKE doesn't exist then the ACK message sent by VICTIM will not get any reply and in the end the

connection times out.

Due to this FAKE and REAL IP reasons, when a person is trying to perform an IP Spoof, he does not get any response from the remote host and has no clue whether he has been successful or not. If he has made any progress or not. You are as good as blind, with no medium through which you could get feedback.

IP Spoofing can be successful only if the computer with the FAKE IP does not reply to the victim and not interrupt the spoofed connection. Take the example of a telephone conversation, you can call up a person ' x ' and pretend to be ' y ' as long as ' y ' does not interrupt the conversation and give the game away.

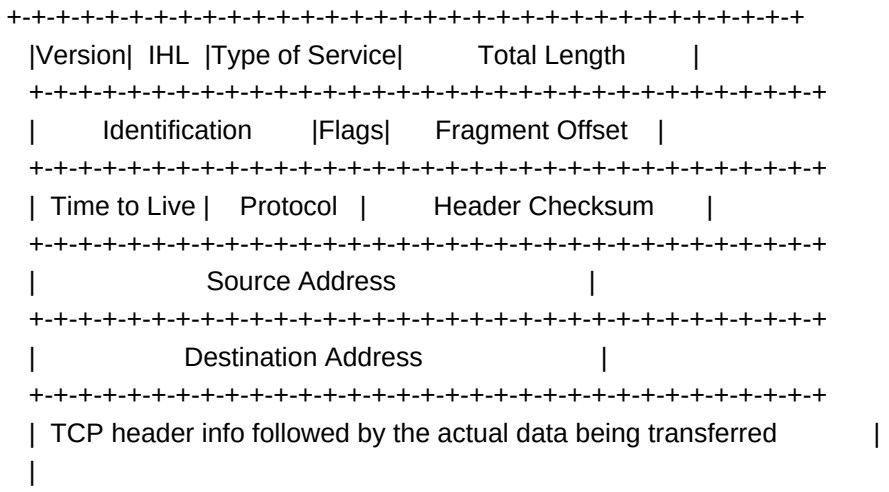
So why would you need to perform IP Spoofing-:

1.) To Pretend that you are some other computer whose IP address is amongst the trusted list of computers on the victim's disk. This way you are exploit the 'r' services and gain access to the network as you are then believed to be from a trusted source.
2.) To Disguise or Mask your IP address so that the victim does not know who you really are and where the data is coming from.

If you ever read the alt.2600 or the alt.hacking newsgroup, you would probably find many postings like "I have Win98, how do I Spoof my IP" or even " I do not know TCP/IP. tell me how to perform IP spoofing". You see the very fact that they are posting such questions and expect to learn how to spoof their IP without even knowing a bit about TCP\IP, confirms the fact that they would not be able to perform IP Spoofing. No I am not saying that asking questions is bad, but you see not knowing something is not so bad, but not knowing something and showing ignorance towards learning it is really, really bad.

You see IP spoofing is a very complex and difficult to perform subject. You need to hog entire TCP/IP and Networking Protocols manuals and need to be able to write C programs which will help you in the Spoofing process. It is amazing how people even think that they can spoof their IP without even knowing what TCP/IP stands for.

You see all packets travelling across the Internet have headers which contain the source and destination IP addresses and port numbers, so that the packet knows where to go and the destination knows where the packet has come from and where to respond. Now the process of Spoofing means to change the source IP address contains by the Header of the packet, in turn fooling the receiver of the Packets into believing that the packet came from somewhere else, which is a fake IP. Now let's again look at the IP Header of a datagram.



Now basically to perform IP spoofing we need to be able to change the value of the field, Source Address. Now to this you need to be able to guess sequence numbers which is quite a sophisticated process and I will try to explain it as clearly as possible. Before we go on, you need to understand the fact the IP spoofing is

not the entire process, it is just a stepping stop in the entire process of fooling the remote host and establishing a trust relationship with the remote host.

So how do these trust relationships take place? Well all of you are encountered with some form of authentication process or the other. Now the Username-Password pair is the most commonly used form of authentication, with which we are very much familiar. Now what happens in the Username-Password form of authentication is that the remote host to which the client is connected to challenges the client by asking the User to type in the Username and Password. So in this form of authentication, the User needs to intervened and the remote host challenges the user to enter the Username and Password which act as a from of authentication.

Now other than the Password-Username form of authentication there is yet another form of authentication most users do not know of. This is the Client IP. In this form of authentication, what happens is that the remote host gets or find out the IP address of the client and compares it with a predefined list of IP's. If the
IP of the client who is trying to establish a connection with the remote host is found in the list of IP's maintained by the host, then it allows the client access to the shell 'without a password' as the identity of
the client has already been authenticated.

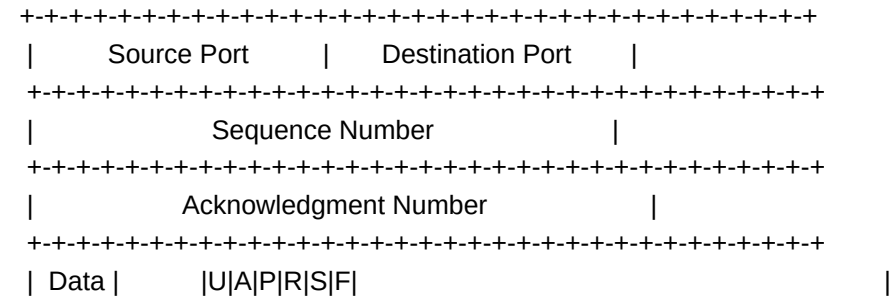
Such kind of rust relationships are common in Unix Systems which have certain 'R services' like rsh , rlogin , rcp which have certain security problems and should be avoided. Despite the threat involved most ISP's in India still keep the ports of the R services open to be exploited by Hackers. You normally establish a Rlogin trust relationship by using the Unix command,

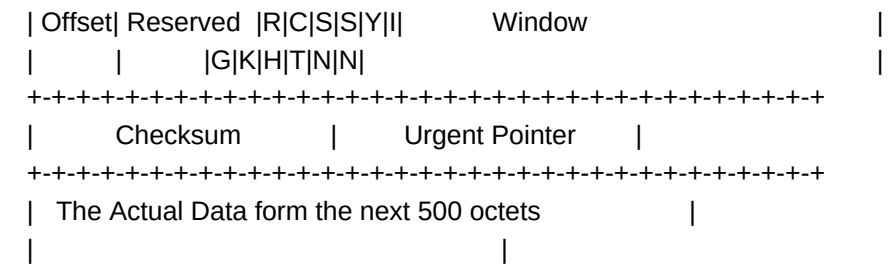
$>rlogin IP address

****
HACKING TRUTH: Well there is definitely a cooler way of establishing a trust relationship with a remote host, using Telnet. The default port numbers at which the R services run are 512, 513,514
****

So how do I spoof my IP? Well in short, to spoof your IP, you need to be able to predict sequence numbers, this will clearer after reading then next few paragraphs.

To understand Sequence Numbers you need to go back to, how the TCP protocol works. You already know that TCP is a reliable protocol and has certain in-built features which have the ability to rearrange, re-
send lost, duplicated or out of sequence data. To make sure that the destination is able to rearrange the datagrams in the correct order, TCP inserts two sequence numbers into each TCP datagram. One Sequence number tells the receiving computer where a particular datagram belongs while the second sequence number says how much data has been received by the sender. Anyway, let's move on, TCP also relies on ACK and NACK messages to ensure that all datagrams have reached the destination error free.
Now we need to reanalyze the TCP Header to understand certain other aspects of sequence numbers and the ACK Number.





You see the TCP Header contains a Sequence Number which actually represents the sequence number of the first byte of that particular TCP segment. A sequence number is a 32 Bit number which is attached to all bytes (data) being exchanged across a Network. The ACK Number Field in the TCP header, actually contains the value of the sequence number which it expects to be the next. Not only that, it also does what it was meant to do, acknowledge data received. Confused? Read it again till you get the hang of it.

When a connection is established, the initial sequence number or ISN is initialized to 1. This ISN number is then incremented by 128,000 every second. There is a certain patter according to which the sequence numbers increment or change which makes then easy to predict.

To successfully perform IP spoofing or in order to predict Sequence Numbers, you need to be running a form of UNIX, as Windows does not provide the users with access to really advanced system stuff. Without a form of Unix IP Spoofing is almost impossible to do.

This text is not the ultimate guide to IP Spoofing and was aimed at only giving you a general outline of the whole process. Sequence number Prediction is really, really sophisticated and difficult to understand, but not impossible to do. However a system administrator can easily save his systems from IP spoofing and this actually makes it quite useless, nonetheless truly exciting. If You really want to learn IP Spoofing I suggest you read IP Spoofing Demystified by daemon9/route/infinity which was a part of Issue 48 of PHRACK magazine, File 14 of 18. Go to the Archive Section of their site, http://www.phrack.com and click on Issue 48.

This brings me to the other purpose people use IP Spoofing, IP Masking. Now to something as simple as mask or hide your IP you do not need to go through the complex procedure of guessing sequence numbers and performing IP Spoofing. There are proxy servers to do that for you. Read the Net Tools chapter for further details.

Port Scanning in Networking Terms

Earlier we learnt what a Port scan is why it is considered to be such a important tool of getting information about the remote host, which in turn can be used to exploit any vulnerabilities and break into the system. We all know how a manual Port Scan works. You launch Telnet and manually Telnet to each Port jotting down information that you think is important. In a manual Port Scan, when you telnet to a port of a remote host, a full three way handshake takes place, which means that a complete TCP connection opens.

The earliest and the oldest version of Port Scanners used the same technique. They connected to each port and established a full three way handshake for a complete TCP connection. The downside of such port scanners was the fact that as a full TCP connection was being established, the system administrator could easily detect that someone is trying to port scan his systems to find a vulnerability. However such port scanning methods also had a bright side, as an actual TCP connection was being established, the port scanning software did not have to build a Fake Internet Protocol Packet. (This IP Packet is used to scan the remote systems.) Such TCP scanners too relied on the three-way TCP handshake to detect if a port is open or not. The Basic process of detecting whether a port is open or not has been described below:

1.) You send a TCP Packet containing the SYN flag to remote host.
2.) Now the remote host checks whether the port is open or not. If the port is open then it replies with a

TCP packet containing both an ACK message confirming that the port is open and a SYN flag. On the other hand if the port is closed then the remote host sends the RST flag which resets the connection, in short closes the connection.
3.) This third phase is optional and involves the sending of an ACK message by the client.

As TCP Scanners were detectable, programmers around the world developed a new kind of port scanner, the SYN Scanner, which did not establish a complete TCP connection. These kinds of port scanners remain undetectable by only sending the first single TCP Packet containing the SYN flag and establishing a half TCP Connection. T understand the working of a SYN or Half SYN Port Scanner simply read its 4 step working-:

1. SYN Port Scanner sends the first TCP packet containing the SYN flag to the remote host.
2. The remote system replies with, either a SYN plus ACK or a RST.
3. When the SYN Port scanner receives one of the above responses, it knows whether the respective port is open or not and whether a daemon is ready listening for connections.

The SYN Port Scanners were undetectable by most normal system port scan detectors, however newer post scan detectors like netstat and also some firewalls can filter out such scans. Another downside to such scanning is that the method in which the scanner makes the IP packet varies from system to system.

UDP Scanning

It is yet another port scanning technique which can be used to scan a UDP port to see if it is listening. To detect an open UDP port, simply send a single UDP Packet to the port. If it is listening, you will get the response, if it is not, then ICMP takes over and displays the error message, " Destination Port Unreachable".

FIN Port Scanners

FIN Port Scanners are my favorite type of port scanners. They send a single packet containg the FIN flag. If the remote host returns a RST flag then the port is closed, if no RST flag is returned, then it is open and listening.

Some port scanners also use the technique of sending a ACK packet and if the Time To Live or ttl of the returning packets is lower than the RST packets received (earlier), or if the windows size is greater than zero, then the port is probably open and listening.

The Following is the code of a supposedly Stealth Port Scanner which appeared in the Phrack Magazine.

```
/*
* scantcp.c
*
* version 1.32
*
* Scans for listening TCP ports by sending packets to them and waiting for
* replies. Relys upon the TCP specs and some TCP implementation bugs found
* when viewing tcpdump logs.
*
* As always, portions recycled (eventually, with some stops) from n00k.c
* (Wow, that little piece of code I wrote long ago still serves as the base
*  interface for newer tools)
*
* Technique:
* 1. Active scanning: not supported - why bother.
```

```
*
* 2. Half-open scanning:
*      a. send SYN
*      b. if reply is SYN|ACK send RST, port is listening
*      c. if reply is RST, port is not listening
*
* 3. Stealth scanning: (works on nearly all systems tested)
*      a. sends FIN
*      b. if RST is returned, not listening.
*      c. otherwise, port is probably listening.
*
* (This bug in many TCP implementations is not limited to FIN only; in fact
*  many other flag combinations will have similar effects. FIN alone was
*  selected because always returns a plain RST when not listening, and the
*  code here was fit to handle RSTs already so it took me like 2 minutes
*  to add this scanning method)
*
* 4. Stealth scanning: (may not work on all systems)
*      a. sends ACK
*      b. waits for RST
*      c. if TTL is low or window is not 0, port is probably listening.
*
* (stealth scanning was created after I watched some tcpdump logs with
*  these symptoms. The low-TTL implementation bug is currently believed
*  to appear on Linux only, the non-zero window on ACK seems to exists on
*  all BSDs.)
*
* CHANGES:
* --------
* 0. (v1.0)
*    - First code, worked but was put aside since I didn't have time nor
*      need to continue developing it.
* 1. (v1.1)
*    - BASE CODE MOSTLY REWRITTEN (the old code wasn't that maintainable)
*    - Added code to actually enforce the usecond-delay without usleep()
*      (replies might be lost if usleep()ing)
* 2. (v1.2)
*    - Added another stealth scanning method (FIN).
*      Tested and passed on:
*      AIX 3
*      AIX 4
*      IRIX 5.3
*      SunOS 4.1.3
*      System V 4.0
*      Linux
*      FreeBSD
*      Solaris
*
*      Tested and failed on:
*      Cisco router with services on ( IOS 11.0)
*
* 3. (v1.21)
*    - Code commented since I intend on abandoning this for a while.
*
```

```
* 4. (v1.3)
*    - Resending for ports that weren't replied for.
*      (took some modifications in the internal structures. this also
*  makes it possible to use non-linear port ranges
*  (say 1-1024 and 6000))
*
* 5. (v1.31)
*    - Flood detection - will slow up the sending rate if not replies are
* recieved for STCP_THRESHOLD consecutive sends. Saves alot of resends
* on easily-flooded networks.
*
* 6. (v1.32)
*      - Multiple port ranges support.
*        The format is: <start-end>|<num>[,<start-end>|<num>,...]
*
*        Examples: 20-26,113
*                  20-100,113-150,6000,6660-6669
*
* PLANNED: (when I have time for this)
* -----------------------------------
* (v2.x) - Multiple flag combination selections, smart algorithm to point
*          out uncommon replies and cross-check them with another flag
*
*/

#define RESOLVE_QUIET

#include <stdio.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_tcp.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <errno.h>
#include "resolve.c"
#include "tcppkt03.c"

#define STCP_VERSION "1.32"
#define STCP_PORT  1234         /* Our local port. */
#define STCP_SENDS 3
#define STCP_THRESHOLD 8
#define STCP_SLOWFACTOR 10

/* GENERAL ROUTINES ---------------------------------------- */

void banner(void)
   {
printf("\nscantcp\n");
printf("version %s\n",STCP_VERSION);
```

```
   }
void usage(const char *progname)
   {
printf("\nusage: \n");
printf("%s <method> <source> <dest> <ports> <udelay> <delay> [sf]\n\n",progname);
      printf("\t<method> : 0: half-open scanning (type 0, SYN)\n");
printf("\t          1: stealth scanning (type 1, FIN)\n");
printf("\t          2: stealth scanning (type 2, ACK)\n");
printf("\t<source> : source address (this host)\n");
printf("\t<dest>   : target to scan\n");
printf("\t<ports>  : ports/and or ranges to scan - eg: 21-30,113,6000\n");
printf("\t<udelay> : microseconds to wait between TCP sends\n");
printf("\t<delay>  : seconds to wait for TCP replies\n");
printf("\t[sf]     : slow-factor in case sends are dectected to be too fast\n\n");
   }
/* OPTION PARSING etc ---------------------------------------- */
unsigned char *dest_name;
unsigned char *spoof_name;
struct sockaddr_in destaddr;
unsigned long dest_addr;
unsigned long spoof_addr;
unsigned long usecdelay;
unsigned      waitdelay;

int slowfactor = STCP_SLOWFACTOR;

struct portrec          /* the port-data structure */
{
  unsigned          n;
  int               state;
  unsigned char     ttl;
  unsigned short int window;
  unsigned long int  seq;
  char              sends;

} *ports;

char *portstr;

unsigned char scanflags;

int done;

int rawsock;         /* socket descriptors */
int tcpsock;

int lastidx = 0;        /* last sent index */
int maxports;                   /* total number of ports */

void timeout(int signum)        /* timeout handler          */
   {       &
```