TCP/IP protocol Possible Attacks

By

Saleh Almanei: almanei@engr.orst.edu

Mohammad Alqattan: alqattan@ece.orst.edu

Rabah Khamis: khamis@ece.orst.edu

Yousaf Hussain: hussain@engr.orst.edu

Oregon State University
Department of Electrical & Computer Engineering

ECE 478/578 Computer and Network Security

Spring Term 2000

Table of contents

<u>Introduction:</u>

The TCP/IP protocols could be considered the most important protocols in the world today. Over the past decade, the Internet has grown from a small network connecting a small community of researchers to its present state - a gigantic global network connecting people of all types. The huge success of the Internet has, for the most part, been quite beneficial. The Internet has evolved from a specialized project to a general-purpose tool. However, the growth of the Internet has created problems with security. The TCP and IP protocols were designed when the Internet was small, and users generally trusted each other. The protocols lack many features that are desirable or needed on an insecure network.

In TCP/IP, there are four layers of protocols that provide the means for a system to deliver data to other systems on the network. Each layer works independently of the other layers and passes the data to either the layer above it or the layer below it. The TCP/IP model defines 4 layers - The Link, Internet, Transport and Application:

NETWORK ACCESS LAYER(Link):

Is the first layer, and it deals with the physical network. It has a protocol for every physical network standard. Functions performed at this level include encapsulating IP datagrams into the frames transmitted by the network. What it does is prepare the data so that it can be operated on by the level above it, the Internet Layer.

INTERNET LAYER:

Is the most important of the TCP/IP protocol suite. Its functions include defining the Internet addressing scheme and learning whether or not to pass it up or pass it on. This is where the fragmentation or reassembly of datagrams occurs. It is connectionless, meaning it does not have to connect to the system that is receiving or sending the data.

HOST-TO-HOST TRANSPORT LAYER:

Is where the TCP resides. TCP provides reliable data delivery by acknowledging that it received the data to the sending host and setting the protocol for checking for errors. TCP is responsible for delivering the data to the correction application at the next layer, the APPLICATION LAYER.

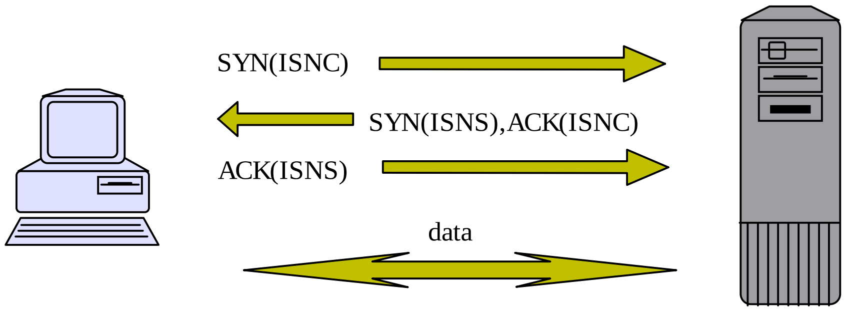
APPLICATION LAYER:

Contains many application protocols such as TELNET, FTP AND SMTP (Simple Mail Transfer Protocol) which delivers electronic mail.

The TCP-header contains about 6 fields with one bit for flags. The ACK bit is used to indicate that the value carried in the acknowledgment field is valid. The RST, SYN and FIN bits are used for connection setup and teardown. When the PSH bit is set, this is an indication that the receiver should pass the data to the upper layer immediately. Finally, the URG bit is used to indicate there is data in this segment that the sending-side upper layer entity has marked as “urgent.” The location of the last byte of this urgent data is indicated by the 16-bit urgent data pointer. TCP must inform the receiving-side upper layer entity when urgent data exists and pass it a pointer to the end of the urgent data. (In practice, the PSH, URG and pointer to urgent data are not used. However, we mention these fields for completeness.)

TCP is used by applications that require reliable data delivery. TCP uses a check-sum with the data sent that the TCP layer at the other end uses to verify that the data arrived undamaged. If the data checks out, the receiving TCP layer sends an acknowledgment back to the sender. If it arrives damaged, the receiver discards it and the sender, after an appropriate time-out period, resends it. But before data is transmitted, the two hosts exchange a three-way handshake. The sender sends a signal telling the receiver it wants to set up a connection and how the data will be sequenced. The receiving host sends a signal back acknowledging receiving the signal. The sending host sends a signal back acknowledging the acknowledgement and begins data transmission. The connection at the end of the data transmission is closed with another 3-way handshake. TCP hands off the data to the IP layer below it. IP breaks the data into packets datagrams and addresses them with the address of the destination and the source. IP is connectionless and does not need the acknowledgment or handshake of the receiving host. The datagram is routed through gateways until it arrives at the destination host. IP passes the datagram to the network access layer, which contains protocols for every physical network standard. This layer formats the data to meet network constraints, encapsulating the IP datagrams into the frames transmitted by the network and mapping the IP addresses to their physical addresses used by the network. At the destination host the data is handled in the reverse order. The network access layer receives the frame and checks the IP address. It either forwards the data to another gateway or sends it to the receiving host. The Internet layer takes the datagrams and begins assembling them back into their original form, passing it up to the transport layer. If it came from an application such as Telnet, FTP or SMTP, TCP checks for errors and passes the data to the correct application.

This paper describes a variety of basic flaws, attacks, in TCP/IP protocols and their implementations, and discusses solutions and work-around to these problems. Those attacks are, Denial Of Service attack, Packet Sniffing, Spoofing attack, TCP sequence number generation, and IP Half Scan. A simple figure explaining the use of TCP will be used as an example:



(Figure 1)

In general, when describing such attacks, our basic assumption is that the attacker has more or less complete control over some machine connected to the Internet. This may be due to flaws in that machine's own protection mechanisms, or it may be because that machine is a microcomputer, and inherently unprotected. Indeed, the attacker may even be a rogue system administrator.

Denial of Service Attack:

In a normal connection, the client sends a message asking the server to authenticate it. The server returns the authentication approval (ACK) to the client. The client acknowledges this approval and is then allowed onto the server.

In a denial of service attack, the client sends several requests to the server, filling it up. All requests have false return addresses and so the server can't find the user when it tries to send the request approval. The server waits, sometimes more than a minute (typically 75 seconds), before closing the connection. When it does close the connection, the attacker sends a new batch of forged requests, and the process begins again--tying up the service indefinitely.

Flaw:

SYN attacks (also known as SYN Flooding or denial of service attacks) take advantage of a flaw in how most hosts implement this three-way handshake which is an important feature of the TCP/IP protocol (figure 1). When Host B receives the SYN request from A, it must keep track of the partially opened connection in a "listen queue" for at least 75 seconds or some other period of time. This is to allow successful connections even with long network delays. The problem with doing this is that many implementations can only keep track of a very limited number of connections. A malicious client can exploit the small size of the listen queue by sending multiple SYN requests to a server, but never replying to the SYN&ACK the server sends back. The server will wait for acknowledgments from the client. As a result, the server’s listen queue is quickly filled up, and it will stop accepting new connections, until a partially opened connection in the queue is completed or times out. This ability to effectively remove a host from the network for at least 75 seconds can be used solely as a denial-of-service attack, or it can be used as a tool to implement other attacks, like IP spoofing. If a malicious client keep sending these phony requests to the server every 75 seconds, then he can keep the server busy and the server keeps denying service to other legitimate requests from other clients.

Because nowadays Internet is not very secure, the damage caused by such an attack can be even more damaging and long lasting. Some attackers, like those who recently attacked the CNN and EBAY web sites, are able to plant small programs that recursively generate requests for establishing connections on other servers and PC's.

Prevention:

One common method of blocking denial of service attacks is to set a filter, or a sniffer, on a network before a stream of information reaches a site server. This filter can guard against attacks by looking for pattern of information or requests. For example, if a pattern comes in frequently, the filter can block messages containing such a pattern and that protects the server from becoming a victim for an attack.

Packet Sniffing Attack:

Packet sniffing is the act of intercepting and reading any or all network traffic that is being transmitted across a shared network communication channel. Packet sniffing is a form of a wire-tap applied to computer networks instead. Unlike telephone circuits, computer networks are shared communication channels. It is simply too expensive to dedicate local loops to the switch (hub) for each pair of communicating computers. Sharing means that computers can receive information that was intended for other machines. To capture the information going over the network is called sniffing.

Flaw:

The most popular way of connecting computers is through Ethernet. Ethernet protocol works by sending packet information to all the hosts on the same circuit. The packet header contains the proper address of the destination machine. Only the machine with the matching address is supposed to accept the packet. A machine that is accepting all packets, no matter what the packet header says, is said to be in promiscuous mode.

Because, in a normal networking environment, account and password information is passed along Ethernet in clear-text, it is not hard for an intruder once they obtain root to put a machine into promiscuous mode and by sniffing, compromise all the machines on the net.

Today's networks are increasingly employing "switch" technology, preventing this technique from being as successful as in the past. It is still useful, though, as it is becoming increasingly easy to install remote sniffing programs on servers and routers, through which a lot of traffic flows.

Packet sniffing is difficult to detect, but can be done. The difficulty of the solution means that in practice, it is rarely done. The popularity of packet sniffing stems from the fact that it sees everything. Typical items sniffed can be:

- SMTP, POP, IMAP traffic: Sniffing one of these allows intruder to read the actual e-mail.
- POP, IMAP, HTTP Basic, Telnet authentication: This allows hacker to read passwords off the wire in clear-text.
- SMB, NFS, FTP traffic: I t allows attacker to reads files of the wire.
- SQL database: It permits the attacker to read financial transactions and credit card information.

Not only can sniffing read information that helps break into a system, it is an intrusion by itself because it reads the very files the attacker is interested in. This technique can be combined with active transmission for even more effective attacks.

Prevention:

Solutions for this problem are readily available. However, most users do not implement these solutions, and are consequently vulnerable to this sort of attack. Authentication schemes such as MD4 and MD5, KERBEROS, and SSH can prevent the clear text transmission of user names and passwords across a network. Public key encryption programs such as PGP can encrypt electronic mail (E-mail) to prevent the contents from being read. Privtool is a neat tool for reading and sending mail using PGP. Unfortunately many organizations do not take advantage of these mechanisms to safeguard their system's integrity because either they do not want to pay for security investments or they do not realize the danger of being vulnerable. Even in cases where organizations do attempt to safeguard there data, faulty algorithm implementations, such as Windows NT's password hashing algorithm, give the impression of more security than actually exists. TELNET packets bound for an NT server, for example, can be intercepted and decrypted by someone knowing the password hashing weakness.

Spoofing Attack:

The term spoofing is used to describe a variety of ways in which hardware and software can be fooled. Although the word spoofing has the connotation of a security threat, but at the same time it can be used for constructive purposes. One such use is as a network management technique to reduce traffic. For example, most LAN protocols send out packets periodically to monitor the status of the network. LANs generally have enough bandwidth to easily absorb these network management packets. When computers are connected to the LAN over wide area network (WAN) connections, however, this added traffic could become a problem. Not only can it strain the bandwidth limits of the WAN connection, but it can also be expensive because many WAN connections incur fees only when they are transmitting data. To reduce this problem, routers and other network devices can be programmed to spoof replies from the remote nodes. Rather than sending the packets to the remote nodes and waiting for a reply, the devices generate their own spoofed replies.

But since our main focus here is on the dark side of spoofing, let's try to see the problem in a broader perspective. As a security threat spoofing is 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 port. To engage in IP spoofing, a hacker must first use a variety of techniques to find an IP address of a trusted port and then modify the packet headers so that it appears that the packets are coming from that port.

Flaw:

As shown in (Figure 1), TCP is a reliable connection-oriented protocol, which requires the completion of a three-way handshake to establish a connection. To implement reliable and unduplicated delivery of data, the TCP protocol uses a sequence based acknowledgment system. During connection establishment each host selects an initial sequence number, which is sent in the first packet of the connection. Each subsequent byte transmitted in the TCP connection is assigned a sequence number. To prevent duplicate or invalid segments from impacting established connections, TCP utilizes a state based model. In a typical client-server application, the client initiates a connection by transmitting a TCP segment to a listening server process. During this state the server acknowledges the clients request setting both the SYN and ACK flags. To complete the three-way handshake the client acknowledges the servers response. To establish a forged TCP session an attacker must have knowledge of or be able to predict the initial sequence number that is selected by the server. An implementation flaw in the Linux kernel allows data to be delivered to the application layer before the handshake has completed.

The existence of this security vulnerability (spoofing), is the result of the combination of three flaws in the Linux TCP/IP implementation. Firstly, Linux only verifies the acknowledgment number of incoming segments if the ACK flag has been set. Linux also queues data from TCP segments without acknowledgment information prior to the completion of the three way handshake but after the initial SYN has been acknowledged by the server. Finally, Linux passes data to the application layer upon the receipt of a packet containing the FIN flag regardless of whether a connection has been established. Together, these flaws allow an attacker to spoof an arbitrary connection and deliver data to an application without the need to predict the servers initial sequence number. According to the standard, there is only one case wherein a correct TCP/IP stack can accept data in a packet that does not have the ACK flag set --- the initial connection-soliciting SYN packet can contain data, but must not have the ACK flag set. In any other case e, a data packet not bearing the ACK flag should be discarded. When a TCP segment carries an ACK flag, it must have a correct acknowledgement sequence number (which is the sequence number of the next byte of data expected from the other side of the connection). TCP packets bearing the ACK flag are verified to ensure that their acknowledgement numbers are correct. Vulnerable Linux kernels accept data segments that do not have the ACK flag set and hence the sequence number is not verified. This allows an attacker to send data over a spoofed connection without knowing the target's current or initial sequence number. Thus, an attacker cannot successfully spoof a TCP transaction to a Linux host without somehow completing the TCP handshake. However, an implementation flaw in some Linux kernels allows an attacker to bypass the TCP handshake entirely, by "prematurely" closing it with a FIN packet. When a FIN packet is received for a connection in SYN_RECEIVE state, Linux behaves as if the connection was in ESTABLISHED state and moves the connection to CLOSE_WAIT state. In the process of doing this, data queued on the connection will be delivered to listening applications. If the ACK flag is not set on the FIN segment, the target's sequence number is not verified in the segment.

<u>Prevention:</u>

TCP/IP spoofing is a dangerous and nearly undetectable security attack that can be carried out on today's Internet. Fortunately there are some protective measures you can take. The best defense is to follow a three-step strategy:

1. Disable JavaScript in your browser so the attacker will be unable to hide the evidence of the attack.
2. Make sure your browser's location line is always visible.
3. Pay attention to the URLs displayed on your browser's location line, making sure they always point to the server you think you are connected to.

This strategy will significantly lower the risk of attack, though you could still be victimized if you are not careful about watching the location line. At present, JavaScript, ActiveX, and Java all tend to facilitate spoofing and other security attacks, so it is recommended that they should be disabled. Doing so might result in loosing some useful functionality, but much of this loss could be recovered by selectively turning on these features when a trusted site is visited that requires them. So far there is no known long term solution to this problem. Changing browsers so they always display the location line would help, although users would still have to be vigilant and know how to recognize rewritten URLs. This is an example of a "trusted path" technique, in the sense that the browser is able to display information for the user without possible interference by untrusted parties. For pages fetched though a secure connection, an improved secure-connection indicator could help. Rather than simply indicating a secure connection, browsers should clearly say who is at the other end of the connection. This information should be displayed in plain language, in a manner intelligible to novice users; it should say something like "Microsoft Inc." rather than “ www.microsoft.com.”

Process Table Attack:

The Process Table Attack is a kind of denial-of-service attack that can be used against numerous network services on a variety of different UNIX systems. The attack is launched against network services which fork() or otherwise allocate a new process for each incoming TCP/IP connection. Although the standard UNIX operating system places limits on the number of processes that any one user may launch, there are no limits on the number of processes that the superuser can create other than the hard limits imposed by the operating system. Since incoming TCP/IP connections are usually handled by servers that run as root, it is possible to completely fill a target machine's process table with multiple instantiations of network servers. Properly executed, this attack prevents any other command from being executed on the target machine.

Flaw:

It has been observed that the UNIX operating system originally contained few defenses to protect it from a denial-of-service attack. This is changing now. With the growth of the Internet, there has been a concerted effort in recent years to strengthen the operating system and its network services to these attacks. Each time a network client makes a connection to a network server, a number of resources on the server are consumed. The most important resources consumed are memory, disk space, and CPU time. Some network services, such as sendmail, now monitor system resources and will not accept incoming network connections if accepting them would place the system in jeopardy. One system resource that has escaped monitoring is the number of processes that are currently running on a computer. Most versions of UNIX will only allow a certain number of processes to be running at one time. Each process takes up a slot in the system's process table. By filling this table, it is possible to prevent the operating system from creating new processes, even when other resources (such as memory, disk space, and CPU time) are widely available. The implementation of many network services leaves them open to a process table attack. That is, an attack in which the attacker fills up the target computer's process table so that no new programs can be executed. The design of some network protocols actually leads the programmer into making these mistakes. An example of such a protocol is the finger protocol (TCP port 79). The protocol follows this sequence:

1. The client makes a connection to the server.
2. The server accepts the connection, and creates a process to service the request.
3. The client sends a single line to the server consisting of the name of the entity that the client wishes to finger.
4. The server performs the necessary database lookup and sends the information back to the client.
5. The server closes the connection.

To launch a process table attack, the client need only open a connection to the server and not send any information. As long as the client holds the connection open, the server's process will occupy a slot in the server's process table. On most computers, finger is launched by inetd. The authors of inetd placed several checks into the program's source code, which must be bypassed in order to initiate a successful process attack. If the inetd receives more than 40 connections to a particular service within 1 minute, that service is disabled for 10 minutes. The purpose of these checks was not to protect the server against a process table attack, but to protect the server against buggy code that might create many connections in rapid-fire sequence.

For a successful process table attack against a computer running inetd and finger, the following sequence may be followed:

1. Open a connection to the target's finger port.
2. Wait for 4 seconds.
3. Repeat steps 1-2.

The attack program is not without technical difficulty. Many systems limit the number TCP connections that may be initiated by a single process. Thus, it may be necessary to launch the attack from multiple processes, perhaps running on multiple computers. Tests have been carried out on a variety of network services on a variety of operating systems. It is believed that the UW imap and sendmail servers are also vulnerable. The UW imap contains no checks for rapid-fire connections. Thus, it is possible to shut down a computer by opening multiple connections to the imap server in rapid succession. With sendmail the situation is reversed. Normally, sendmail will not accept connections after the system load has jumped above a predefined level. Thus, to initiate a successful sendmail attack it is necessary to open the connections very slowly, so that the process table keeps growing in size while the system load remains more or less constant.

A variety of problems on BSD-based machines being used as the attacker have been observed. When the target machine freezes or crashes, the attacker machine sometimes crashes as well. Apparently the TCP stack does not gracefully handle hundreds of connections to the same port on the same machine simultaneously going into the FIN_ WAIT_2 state. There are various ways of this attack:

1. Use IP spoofing so that the incoming connections appear to come from many different locations on the Internet. This makes tracking considerably harder to do.
2. Begin the attack by sending 50 requests in rapid fire to the telnet, rlogin and rsh ports on the target machine. This will cause inetd to shut down those services on the target machine, which will deny administrative access during the attack.
3. Instead of initiating a new connection every 4 seconds, initiate one every minute or so. The attack slowly builds, making it more difficult to detect on packet traces.

Prevention:

Following are some of the suggestions that can help defending against the attack:

1. Programs such as inetd should check to see the number of free slots in the process table before accepting new connections. If there is less than a predefined number of free slots, new connections should be accepted.

2. Alternatively, if there are more than a preset number of network daemons for the service running, incoming requests should be queued rather than serviced.
3. Network services (such as finger) should implement timeouts. For example, the statement alarm (30) could be inserted into the finger daemon source code so that the program would stop running after 30 seconds of execution.

TCP sequence number generation Attack:

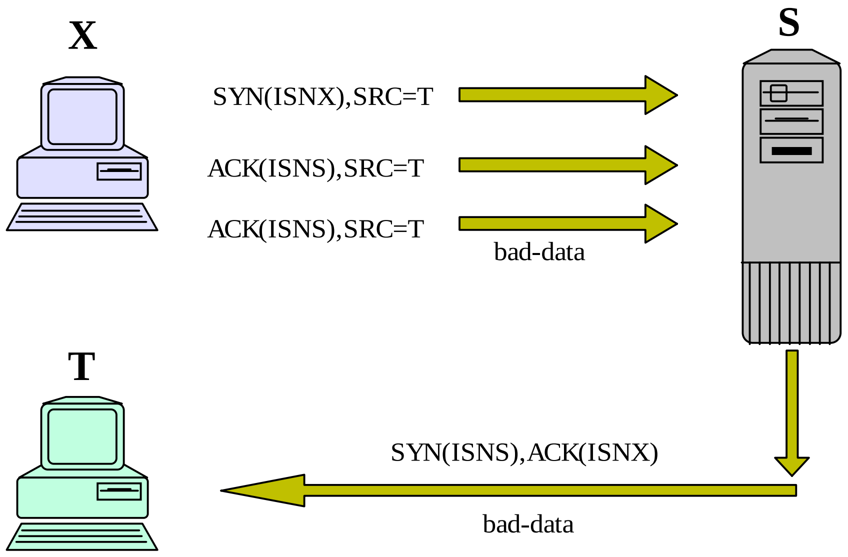
TCP is run over an unreliable IP, and IP does not guarantee delivery of packets, and does not necessarily deliver the packets in sequence, a Delayed packet showing up is not uncommon. Given these conditions, TCP does not make any assumptions about the underlying network. In order to identify packets and ensure that they are delivered to the application layer in order, the sequence number is an important component of every TCP segment. Every byte of data that TCP sends is given a sequence number. By doing this, the sender and receiver can verify whether the data was delivered correctly. They can also determine whether data was dropped, possibly because of loss in transit. Both the sender and the receiver exchange initial sequence numbers (ISN) during the connection setup phase. After a successful initial handshake, both the sender and the receiver know the sequence numbers that they have to use for communication. Since TCP allows for delayed segments, it must accept segments that are out of sequence, but within certain bounds, known as the receiver window size. The receiver window size is also exchanged during the initial handshake. TCP will discard all segments that do not have a sequence number within the computed bounds.

The TCP sequence number is a 32-bit counter. In order to distinguish between different connections between the same sender and receiver, it is important that the sequence numbers do not start at 0 or any other fixed number each time a connection is opened. Hence, it is important that the first byte of data from the sender to the receiver is using a random sequence number. Current implementations increment the sequence number by a finite amount every second.

Flaw:

One of the more fascinating security holes is the use of TCP Sequence Number prediction to construct a TCP packet sequence without ever receiving any responses from the server. This will allow the attacker to spoof a trusted host on a local network. The normal TCP connection establishment sequence involves a 3-way handshake. The client selects and transmits an initial sequence number ISNC, the server acknowledges it and sends its own sequence number ISNS, and the client acknowledges that. Following those three messages, data transmission may take place. (The exchange was shown in figure 1 in t he introduction.)

That is, for a conversation to take place, C must first hear ISNS, a more or less random number. Suppose, though, that there was a way for an intruder X to predict ISNS. In that case, it could send the following sequence to impersonate trusted host T:
X→S:SYN(ISNX),SRC=T, S → T:SYN(ISNS),ACK(ISNX),X→S:ACK(ISNS),SRC=T,
X →S:ACK(ISNS), SRC=T, bad-data ;



Even though the message S→T does not go to X, X was able to know its contents, and hence could send data. If X were to perform this attack on a connection that allows command execution malicious commands could be executed.

Prevention:

If we could prevent people from being able to guess the next sequence number used for a connection based on the previous sequence number used in a connection, we would be able to eliminate this kind of attack. The first thought might be to Incrementing the ISN counter more often; however, this will not help. An attacker may still be able to approximately predict the ISN. By repeatedly guessing, it is likely that the attacker will establish a connection with the correct ISN within a few hours. As long as there is possibilities this solution is not safe and secure.

A most likely solution is to deal with the actual counter representation inside the TCP specification. In another word, we can implement a cipher mechanism to protect the sequence number guessing. DES could be used in place of a random number generator, since it is well studied, and the encryption of an incrementing counter are quite varied and difficult to predict. Of course, all methods of generating pseudo random numbers are subject to analysis; if nothing else, once the seed is discovered, the whole sequence is known. Whatever we choose as a solution, we should certainly avoid using a simple, easily predicted sequence number generator.

IP Half Scan:

SYN-scanning or “IP Half Scan” implies that a full TCP connection is never established. The process of establishing a TCP connection is three phased:

> The originating party first sends a TCP packet with the SYN flag on, then the target party sends a TCP packet with the flags SYN and ACK on if the port is open, or, if the port is closed, the target party resets the connection with the RST flag. The third phase of the negotiation is when the originating party sends a final TCP packet with the ACK flag on (all these packets, of course, have the corresponding sequence numbers, ack numbers, etc). The connection is now open.

A SYN-scanner only sends the first packet in the three-way-handshake, the SYN packet, and waits for the SYN|ACK or a RST. When it receives one of the two it knows whether or not the port is listening.

Flaw:

An attacker can determine which services respond on specific ports, often without causing the target computer to log a connection or to detect the attempt. During a normal TCP connection, as mentioned above, the source initiates the connection by sending a SYN packet to a port on the destination machine. If a service is listening on that port, the service will respond with a SYN/ACK. The client initiating the connection then responds to the SYN/ACK with an ACK, which completes the TCP connection, and quite often logs information about the connection being established. During a Stealth Scan, instead of sending an ACK in response to the SYN/ACK after the client has determined a service is listening on that port, the client sends a RST packet, which aborts the connection. The source machine is then able to determine that a service is listening on a port without actually establishing the connection. In most cases, the failure to establish the connection does not result in the connection being logged. This mechanism affects any platform that use TCP/IP protocol. The possibilities of threat gain from the first step attackers usually do which is collecting info about the victim machine.

Prevention:

Using a firewall that understand the state of TCP connections and rejects stealth scan packets. Stateful Inspections and Proxy firewalls will defeat IP half scan flaw. This is not a major problem as others discussed in this paper. The solution is widely available in the market these days.

Conclusion:

TCP/IP, as it exists today, has a general lack of security. This has enabled the malicious attackers to wage different kinds of attacks such as Denial of Service Attack, Packet Sniffing Attack, Spoofing Attack, Process Table Attack, TCP Sequence Number Generation, and IP Half Scan, exploiting TCP/IP weaknesses. As we have mentioned in the Introduction, TCP/IP is presently the most widely used network protocol. This shows how important it is to focus our attention on how to make this protocol safe from the attacks as mentioned above. Fixing some of these flaws today is possible with add-ons like TCP Wrappers, Kerberos, and SKIP, but these fixes are not always in widespread, everyday use. For example a client host may implement them, but the host that the client host want to communicate with may not. Thus, most communication on today's Internet is still unsecured. Although some of the preventive measures mentioned above for each individual kind of attack, may make the protocol more secure, persistent attacker would never stop looking for new security holes and exploit them. Unfortunately this is not only true for TCP/IP, but for all other security systems in general. Migrating to a new protocol suite, such as IPSec which ensures confidentiality, integrity, and authenticity of data communications across a public network. IPSec provides a necessary component of a standards-based, flexible solution for deploying a network-wide security policy. In this way even if an attacker finds a way to access the information that is being exchanged between two parties it would not do him any good. Because the information that he has gathered is in an encrypted form and he has no way to decrypt it. For this information to be of any use for him, he has to have the decryption keys for both parties. Although it would be appropriate to elaborate more on IPSec, it still is beyond the scope of this paper.

Reference:

1. Computer Networking; A Top-Down Approach Featuring the Internet; James F. Kurose and Keith W. Ross; http://www.seas.upenn.edu/~ross/book/Contents.htm

2. http://gaia.cs.umass.edu/kurose/transport/segment.html; Copyright Keith W. Ross and James F. Kurose 1996-2000.

3. http://www.insecure.org.

4. A simple active attack against TCP; Laurent Joncheray. PDF Document; 1995.

5. A Cryptographic Evaluation of Ipsec, Niels Ferguson and Bruce Schneier, Counterpane Internet Security, Inc., http://www.counterpane.com.

6. Security Problems in the TCP/IP Protocol Suite, *S.M. Bellovin*, Reprinted from Computer Communication Review, Vol. 19, No. 2, pp. 32-48, April 1989. http://www.ja.net/CERT/Bellovin/TCP-IP_Security_Problems.html

7. TCP/IP Security, CHRIS CHAMBERS, JUSTIN DOLSKE, and JAYARAMAN IYER. Department of Computer and Information Science, Ohio State University. http://www.cis.ohio-state.edu/~dolske/gradwork/cis694q/

8. Phrack Magazine, Volume Seven, Issue Forty-Eight, File 14 of 18. http://www.2600.com/phrack/p48-14.html

9. Data Flow on TCP-IP Networks, http://tfbbs.tvinet.com/amitcp/Data_Flow_on_TCP-IP_Networks.html