Internet Security Attacks at the Basic Levels

Marco de Vivo
mdevivo@reacciun.ve
Apartado Postal 68274
Caracas, Venezuela.

Gabriela O. de Vivo
gdevivo@reacciun.ve

Germinal Isern
isern@reacciun.ve

GIRAS U.C.V.

Abstract

The Internet put the rest of the world at the reach of our computers. In the same way it also made our computers reachable by the rest of the world. Good news and bad news!. Over the last decade, the Internet has been subject to widespread security attacks. Besides the classical terms, new ones had to be found in order to designate a large collection of threats: *Worms*, *break-ins*, *hackers*, *crackers*, *hijacking*, *phrackers*, *spoofing*, *man-in-the-middle*, *password-sniffing*, *denial-of-service*, and so on.

Since the Internet was born of academic efforts to share information, it never strove for high security measures. In fact in some of its components, security was consciously traded for easiness in sharing. Although the advent of *electronic commerce* has pushed for "real security" in the Internet, there is yet a huge amount of users (including scientists) very vulnerable to attacks, mostly because they are not aware of the nature (and ease) of the attacks and still believe that a "good" password is all they need to be concerned about.

We wrote this paper aiming for a better understanding of the subject. In the paper we report some of the major actual known attacks. Besides the description of each attack (the *what*), we also discuss the way they are carried on (the *how*) and, when possible, the related means of prevention, detection and/or defense.

Keywords: TCP/IP, Client-Server, Ethernet, Sniffing, One-Time Password, Kerberos, ICMP, Ping, Covert Channel, Spoofing, RIP, DNS, Denial of Service, Hijacking.

1. Introduction.

Although the terms *Protection* and *Security* are used interchangeably in the related literature, it is worth noting that, strictly speaking, they designate different subjects.

Protection [1] refers to a **mechanism** for controlling the access of programs or users to a set of resources. *Security*, on the other hand, refers to a measure of confidence in the **policies** defined for the correct (as planned) management of an environment. Thus, as strange as it may appear, it is possible to have a weak security **policy** even when using a very sound **mechanism** of protection. As an example, imagine a user who systematically selects as her password (a strong protection mechanism) the same as her login name (a very weak security policy indeed!!).

The cases analyzed in this paper are mainly related to faulty protection mechanisms: bad design, wrong implementations and poor integration are at the base of most of the attacks, including those directed to exploit wrong security decisions.

Active vs. ***Passive*** attacks:

A successful attack is usually achieved through several active or passive sub-attacks. In active attacks, the intruder **interacts** with the target system and tries to influence its behavior. These attacks involve some modification of the data stream or the creation of a false stream. In passive attacks, the intruder just collects information related to (usually *proceeding from*) the selected victim, and, normally[1], no alteration of the data is involved. Passive attacks are in the nature of eavesdropping on, or monitoring of, transmissions (e.g., sniffing devices and keystrokes capturing programs).

Fundamental attacks:

Almost all attack strategies end with what we will call a fundamental attack.

The goal of a fundamental attack is to allow an intruder gaining entrance to the victim's system either as a regular user or (mostly) as the *superuser*. There are basically three forms of fundamental attacks:

- Password guessing or capturing. Password guessing is also known as ***password cracking*** [4].
- Exploiting security flaws (***security holes***).
- Exploiting bugs in Operating System and Network software (***security holes***).

We won't cover password guessing or classical security holes because they are thoroughly studied and well documented elsewhere. Instead, a specific passive attack (***sniffing***) and several chosen **active** attacks will be analyzed in detail.

[1] Data streams protected by *Quantum Cryptography* cannot be eavesdropped unnoticeably [2, 3].

The presentation of the selected attacks is arranged nearly following the TCP/IP layered approach.
We wrote this paper in an effort to provide information about how intrusions occur. Unfortunately users and system administrators are often unaware of the extent and dangerousness of the threats beyond the most well known banal attacks. Moreover, even experienced computer advisers, usually get into trouble when asked with questions of the pattern: '...but how, exactly, do ***hackers*** break into systems?...'.
We really hope that the paper will **help** readers build a solid understanding of the above mentioned issues, as well as better estimating their own risks and security requirements.

2. About TCP/IP.

In order to fully understand the issues discussed in this paper, some basic knowledge of TCP/IP is needed. In this section several fundamental concepts and definitions, related to TCP/IP, are presented. Additional information can be found in **[5, 6, 7]**. Expert readers are invited to skip to the next section.
TCP/IP is a *protocol suite* conceived to allow computers of different characteristics (software and hardware), to communicate with each other. An *internet* is a collection of networks that all use the same protocol suite. The Internet (note uppercase **I**) is based on TCP/IP.
A protocol suite is normally the combination of different protocols at various layers. TCP/IP is usually considered a 4-layer system, as shown in **Figure 1**.

• The ***host-to-network*** layer: This layer, also called *link layer*, *data link layer* or *network interface layer*, handles the details of the communication media. It is related to issues like device drivers, *Ethernet*, *token ring*, interface cards, etc.

• The ***network*** layer: The network (also called *internet*) layer is the glue that holds the whole architecture together. It deals with the movement of packets around the network, and defines an official packet format and protocol called **IP (Internet Protocol)**.
IP offers a connectionless, unreliable service, called *datagram service*. This means it does it best job of moving packets from sources to final destinations, but there are no guarantees. IP also specifies an addressing scheme based on the so called *Internet addresses* or *IP addresses*. Every interface on an internet must have a **unique** IP address.
Network layer is also related to issues like packet *routing* and congestion avoidance.

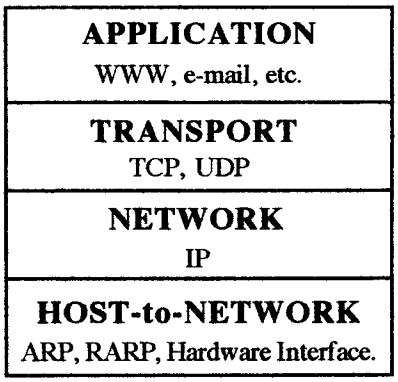

Figure 1. TCP/IP protocol suite.

• The ***transport*** layer: The transport layer is designed to provide a data **stream** between communicating applications on source and destination *hosts* (computers). When an application running on the source host needs to communicate with its mate on the destination host, it just gives the message to the transport layer's software, which in turn uses IP to send it to the target host in form of data units (packets) known as *datagrams*. This layer also deals with *flow control*, *congestion control*, and with the creation of *end points*.
Two end-to-end protocols have been defined in this layer: ***TCP*** and ***UDP***.
TCP provides a **reliable** flow of data to communicating applications. It does so, even though the service it uses to forward packets (IP) is unreliable. Reliability is achieved by the use of timers, counters, acknowledgments and re-transmission.
UDP, on the other hand, just provides the primary mechanism that application programs use to send datagrams (units of information) to other application programs. The same as TCP does, it uses the underlying IP protocol to transport a message from one machine to another, but unlike TCP it provides the same **unreliable**, connectionless datagram delivery semantics as IP. It does not use acknowledgments, nor orders incoming messages, and it does not control the rate at which information flows between the machines either. Thus UDP messages can be lost, duplicated or arrive out of order. Any desired reliability must be added by the application layer. Nevertheless, UDP is a better interface than IP to applications because it adds the ability to distinguish among multiple destinations within a given host computer.

• The ***application*** layer: This layer contains all the higher level protocols. Almost every TCP/IP implementation provides many common applications:
Telnet, a virtual terminal protocol for remote login.
FTP, for the transferring of data between machines.

SMTP, for electronic mail.
It is worth noting that most networking applications are written as two complementary programs: The *client* and the *server*. The purpose of the application is for the server to provide some defined service for clients.

• ***Routers***: Since an internet is a collection of networks, some special-purpose hardware is needed for interconnecting them: *routers* (some computers can act as routers also). When an IP packet is sent from a source host, it usually does not reach directly the target host. Instead, it is handled by several intermediate routers running IP software. Each router, in turn, forwards the packet (following the *routing* information stored in it) until the destination host is reached. So in most cases, packets will require multiple hops to make the journey.
The source and destination hosts are called *end systems*. The application layer and the transport layer use *end-to-end* protocols since they are needed only on the end systems. By contrast IP is a *hop-by-hop* protocol because it is used (see **Figure 2**) on the two end systems and **every** *intermediate system* (router).

• **IP *addresses*** and ***DNS***: Each machine (hosts and routers) on an internet must have at least a distinctly different address so that information destined for it can be successfully delivered. This address scheme is controlled by IP. An IP address is 32 bits long (in IP version 4) and consists of two parts: the network portion (used to describe the network on which the host resides) and the host portion (used to identify the particular host). Those machines connected to multiple networks have a different IP address on each network.
IP addresses are usually written in *dotted decimal notation*. In this format each of the 4 bytes is written in decimal from 0 to 255. Thus the hexadecimal address C1C103B0 is represented as 193.193.3.176.
When the IP on a source host receives a unit of data to be forwarded, it adds (as is done at each layer) information to the data by prefixing a *header* to it. This header (IP *header*) includes the IP address of the source host and the IP address of the destination host. In this way, every intermediate system knows exactly the final destination of the *packet* (needed for routing) and the address of the originating host (needed when reporting problems with the packet).
Now, since IP addresses can be difficult to remember, each device is generally assigned a host name (ASCII string). Nevertheless, as IP itself only understands binary addresses, some mechanism is required to convert the ASCII strings to network (IP) addresses and vice versa (when needed). DNS, the *Domain Name System*, is a distributed database implementing a hierarchical, domain-based naming scheme, and used by TCP/IP applications to map between host names and IP addresses. An example will help to understand the whole mechanism: Suppose that hosts **A** and **B** in **Figure 2** are assigned the IP addresses 191.191.3.120 and 193.193.3.176 respectively. Suppose also that their host names are: hosta.net1.com and hostb.net2.com. Now, if a client application running on host **A** (hosta.net1.com) needs to use a server application on host **B** (hostb.net2.com), it must use the DNS first to obtain the IP address of host **B**. This is done as follows: The client application invokes a special program (also running on host **A**, of course) called *the resolver* and passes to it the string 'hostb.net2.com' as an argument. Next, the resolver search its related files for the IP address of a *name server*, and, once found, uses it to query the name server about 'hostb.net2.com'. The name server searches its portion of the data base, and, if necessary, queries other name servers until the desired IP address (193.193.3.176) is obtained. This address is sent back to the resolver which in turn returns it to the invoking client. At this point the connection can be established.

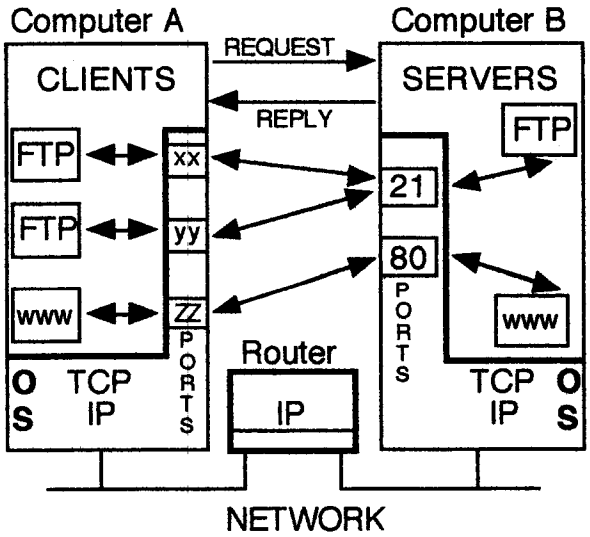

Figure 2. Two hosts connected using TCP/IP.

• ***Ports*** and ***Sockets***: As shown in **Figure 2**, many different applications can be using TCP or UDP at any one time. Several instances of a client on host **A** could be interacting with the same server application on host **B**, and several independent client-server sessions between the two hosts could be running simultaneously as well. To achieve this multiplexing, TCP (UDP) defines the concept of *protocol port*: A *protocol port* is an abstract destination point identified by a 16-bit positive integer (*port number*).
When TCP (UDP) receives from the application layer

data to be transmitted, it adds information to the data by prefixing a *header* to it. This header (TCP or UDP *header*) includes the number of the *destination port* on the machine to which the data is sent, as well as the *source port* number (optional in UDP) on the source machine to which replies should be addressed.
Now, as every application using TCP must be associated with a port number, to communicate with a remote application (foreign port) a sender needs to know **both** the IP address of the destination machine **and** the port number assigned to the application within that machine. The pair (IP address, port number) is called a *socket* and represents an end point of a TCP connection. To obtain TCP service, a connection must be explicitly established between a socket on the sending machine and a socket on the receiving machine. TCP connections are thus identified by its two end points, that is (*socket1*, *socket2*).

3. Attacks related to the link layer.

3.1. Sniffing.

Perhaps the best known source of general information about ***sniffers*** is the **ISS Sniffer FAQ [8]**. Let's examine some of the items treated in that FAQ[2] (**F**requently **A**sked **Q**uestions):

What is sniffing:
Sniffing is the use of a network interface to receive data not intended for the machine in which the interface resides **[9]**. A variety of types of machines need to have this capability. A ***bridge***, for example, usually has **two** network interfaces that normally receive all ***frames*** traveling on the media on one interface and retransmit **some** of these frames on the other interface.

Computer networks are often based on 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. Hence, this kind of networks is particularly suitable for sniffing.
Ethernet is a very popular way of connecting computers through shared communication channels. Ethernet protocol works by sending packet information (frames) to all the hosts on the same circuit. The frame header contains the proper address of the destination machine. Only the machine with the matching address is supposed to accept the frame. A machine (more precisely, a network interface) that is accepting all frames, no matter what the frame 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 intruders, once they obtain *root* (superuser's privileges), to put a machine's network interface into promiscuous mode and by sniffing, compromise all the machines on the net. Besides, sniffing leads to loss of privacy of several kinds of information like financial account numbers, private data (e.g., e-mails), and low-level protocol information (e.g., IP addresses and TCP sequence numbers) among others.

Where can sniffers be found:
Even though sniffers are among the main causes of mass break-ins on the Internet today, they are also invaluable tools for network troubleshooting. For this reason there are several implementations widely available (as shareware and freeware) through the Internet:

TCPDump	ftp://ftp.ee.lbl.gov
EthDump	ftp://ftp.germany.eu.net /pub/networking/inet/ethernet/
Packetman, *Interman*, *Etherman*, *Loadman*, *NetMan*	ftp://ftp.cs.curtin.edu.au /pub/netman/

All these sniffers are provided to assist in the debugging of network problems. They can also be helpful in the study of protocols and in the gathering of (network related) statistical data. Whatever their use, access to them should be restricted to system administrators as well as to selected personnel.

Detecting sniffing attacks:
When a dedicated device (in contrast to a program running on a network's known host) is used for sniffing, detection requires physically checking all the ethernet connections. Otherwise, administrators must check (when possible) for interfaces working in promiscuous mode (it can be done with the help of commands like ***ifconfig*** and programs like ***cpm***). If sniffing is enabled by linking it into the kernel, there are several commands that could also be helpful (e.g., ***pfstat*** and ***pfconfig***).
Often a sniffer log becomes so large that the file space is all used up, besides, on a high volume network a sniffer will create a large load on the machine. These

[2] Although the following discussion is mostly connected with *ethernet*, this kind of attack can be directed to other 'multiple access' protocols (and even to non-broadcast ones) as well.

facts can also lead to the discovery of sniffers.
Regarding PC's with non-Unix systems, observe that except for special cases (e.g., ***Java, CGI programs***, etc.), command execution is not allowed but from the console, therefore remote intruders can not turn a PC machine into a sniffer without inside assistance.

Preventing and neutralizing sniffing attacks:
• *Network segmentation*: A network segment consists of a set of machines sharing low-level devices and wiring and seeing the same set of data on their network interfaces. Repeaters and passive hubs do not limit the flow of data arriving to any of its interfaces. They just copy the incoming bits and retransmit them to the wires on the other interfaces. ***Switches, active*** hubs, and ***bridges***, however, do limit the flow of data, thus allowing the prevention of sniffing on untrustworthy machines.
• *Encryption*: The use of encryption renders data useless to intruders. There are several related packages available. However, this solution is neither universal nor absolute. In fact, there is a tradeoff between how much information is encrypted and how standard the solution is. If too much of the flowing information is encrypted (e.g., even protocol information) then, proprietary networking protocols must be used (thus precluding standard internetworking). If, by contrast, only application level encryption is used, then a significant amount of sensible data (mostly related to network and operating system protocols) is still available to intruders, nevertheless, the two most popular encryption-based security solutions ***PGP*** [10] and ***SSL*** [11] are both designed to guarantee the secrecy, integrity and authenticity of data related to the **application level** only (e.g., e-mails, transactions, etc.).
• *Special administration of account information*: ***S/key*** [12] and other *one-time password* technology make sniffing **account** information almost useless. Passwords never go over the network but rather are used to create (on both connecting sides) matching strings of bytes.
Usually, the server presents a challenge to the connecting user (or client) who, using the challenge information and the real password, either calculates or selects (from a previously defined list) a new string and sends it back to the server. The string is then entered into the server's comparing algorithm, and if a match is obtained, the connection is allowed to continue. Neither challenges nor strings are used twice.
A well-known alternative to one-time passwords is ***Kerberos*** [13]. Kerberos is a system that allows workstations to authenticate themselves to services running on servers without ever sending a password in clear text over the network. Kerberos is based on the use of ***authentication servers***[3]. An authentication server (AS) knows the passwords of all users and stores these in a centralized database. In addition an AS shares a unique secret key with each server.
The protocol is roughly as follows:
1) When a service is needed, the client sends a request to the authentication server.
2) The AS uses the known client's password and the secret key related to the service to create an encrypted message containing an also (differently) encrypted *service-granting ticket*.
3) The message is sent back to the client which decrypts it, extracts the ticket, and presents it to a suitable server.
4) The server tries to decrypt and authenticate the ticket. If succeeds, the requested service is granted.
• *Non-promiscuous interfaces*: Installing interface cards that do not support promiscuous mode, will prevent PC's (usually IBM compatibles) from sniffing.

4. Attacks related to the network and transport layers.

4.1. ICMP Tunneling.

Firewalls, Ping, and ICMP:
Roughly speaking a ***firewall*** is a computer, software, or both, used to restrict and monitor usage of a computer or network. Firewalls are normally used to control the interface between a subnetwork and the Internet.
The Internet Control Message Protocol (***ICMP***) is an adjunct to the IP layer. It is a connectionless protocol used to carry error or control messages between the IP software on one machine (host or router) and the IP software on another. ICMP packets are encapsulated inside IP datagrams. Although each ICMP message has its own format, they all begin with the same three fields (first 4 bytes of the header): TYPE, CODE, and CHECKSUM. There are 15 different types of ICMP messages.
The ***ping*** command sends an ICMP *echo request* message (type 8) to a specified destination. Any machine that receives an *echo request*, formulates an ICMP *echo reply* message (type 0) and returns it to the original sender. Both the request and the reply can include an *optional data* field. Thus ping can be used to test whether a destination is reachable and responding.

[3] Strictly speaking, Kerberos uses a special server program for *key distribution*. The program performs two logical functions: *authentication server* and *ticket-granting server*. In this basic description we are using the term *authentication server* to designate both logical servers as a whole.

Covert channels:
Since ping traffic is ubiquitous to almost every TCP/IP based network and subnetwork, many firewalls and networks consider this traffic to be benign and will allow it to pass through, unmolested. However, that practice can be insecure. Ignoring the obvious threat of a denial-of-service attack, use of ping traffic can open up covert channels through the networks in which it is allowed.
Remember that ICMP echo packets also have the option to include a data section. Although the payload is often timing information, there is no check by any device as to the content of the data. So, as it turns out, this amount of data can also be arbitrary in content as well. Therein lies the covert channel.
A tool exploiting this covert channel **[14]** can be used as a backdoor into a system by providing an unavowed method of getting commands executed on a target machine. It can be used as a way to clandestinely collect information from a machine. It can be used as a covert method of user-machine or user-user communication.

Detection and prevention:
If ICMP *echo traffic* is allowed, then this channel exists. Even with extensive *firewalling* and packet-filtering mechanisms in place, this channel can go completely undetected for the duration of its existence.
A surplus of ICMP echo reply packets with a garbled payload can be ready indication that the channel is in use, but since these packets are usually not monitored, some triggering event must happen first.
Restricting ICMP echo traffic to be accepted from trusted hosts is useless with a connectionless protocol such as ICMP. Forged traffic (with ***spoofed*** IP addresses) containing hidden data in the echo request messages can still reach the target host which in turn will send legitimate ICMP echo reply messages to the spoofed host (where they will be dropped silently). While the possibility exists for a smart packet filter to check the payload field and ensure that it **only** contains legal information, such a filter for ICMP is not in wide usage, and could still be open to deceiving.
The only sure way to destroy this channel is to deny **all** ICMP echo traffic into the (to be protected) network, a very impractical requirement, though.

5. Attacks related to the network and transport layers.

5.1. IP Spoofing.

The term *trusted host* was coined by the developers of the UNIX networking software. If one host extends trust to another host, then any user who has the same *username* (login name) on both machines can log in (or execute *remote* commands) from the trusted host to the trusting one **without presenting a password**. Trust can also be extended to different users from selected hosts, and eventually to **any** user from **any** host!!. The trusted usernames and hostnames are maintained in two types of special files (UNIX): ***.rhosts*** and ***hosts.equiv***. Any authorized user of a host **A**, can create in **its own** directory (on **A**) a file named .rhosts containing the combinations of username and hostname that may connect to **its** account on **A**. By contrast, only **one** hosts.equiv file (/etc/hosts.equiv) may exist on each host. If present, it contains a list of trusted hosts. Any user of any host in the list may access an account with the same username on the trusting host without presenting a password.
It is worth noting that besides ***rlogin*** (the login command from a **remote** host), several other commands use the *trusted host* scheme (e.g., ***rcp***, ***rdist***, ***rsh***.) They are known collectively as ***r**** commands.
Although *trusted host* is a very useful and convenient scheme, its actual implementation is quite vulnerable to attacks because an authentication mechanism based solely on IP addresses is used. In fact, when any of the r* commands is invoked from a remote machine, the receiving host checks if the IP address of the sender matches an authorized (trusted) host. If so, the command is executed. Otherwise, either permission for execution is denied or a **password** (and a **login** if the originating user is not equivalent to the remote user) is (are) prompted for on the remote machine.
IP Spoofing attacks exploit this weak form of authentication. In this type of attack, an intruder masquerades her host **evil.com** as **friend.com** a machine (usually internal) trusted by the host **target.com**. The intruder does this by substituting the IP address of the trusted machine, **friend.com**, for the IP address of her host, **evil.com**, in all of the outgoing packets. The machine being attacked, **target.com**, then believes that the intruder is, in fact, the machine that it trusts, **friend.com**, and gives it access.

5.1.1. ARP Spoofing[4].

ARP, the Address Resolution Protocol, provides a mapping between two different forms of addresses: 32-bit IP (virtual) addresses used by the network layer, and whatever type of physical address (e.g., Ethernet) the associated data link uses. An Ethernet *dynamic* ARP

[4] ARP is considered as part of the link layer. However, we included ARP spoofing in this section because of its strong relation with attacks to the IP based trusting mechanism.

(which has become a TCP/IP Internet protocol standard) is specified in RFC 826 [15].
The protocol works roughly as follows: When an IP datagram is to be sent, either the destination host (the one owning the **target** IP address) is in the same physical subnet, or a gateway must be used as the first hop. In any case, a physical Ethernet address is needed (the destination host Ethernet address or the gateway Ethernet address) to send the Ethernet frame encapsulating the IP packet. So, the ***ARP cache*** of the source machine is consulted looking for an entry associating the target (host or gateway) IP with an Ethernet address. If there is a *miss*, a special Ethernet frame (an ***ARP request***) is *broadcasted* to every host on the network. The ARP request contains the source Ethernet and IP addresses, and the IP address of the target. Every machine receiving the request can extract the sender's IP-to-physical address binding, and update its cache. Additionally, the destination machine replies with an ***ARP reply***, containing its IP address and the corresponding hardware address. When the ARP reply is received by the source machine, its cache is updated and the datagram that forced the ARP request-reply to be exchanged can finally be sent.
Since the entries in an ARP cache usually expire after a few minutes, several attacks, aiming to forge or tamper with the IP-to-hardware address associations, are possible. An attacker can simply use a machine assigned the same IP address as a machine currently not working. The machine to be impersonated can be turned off, or disconnected from the network, or simply having its legitimate IP address changed by the attacker. Then, after waiting a few minutes for the expiration of the original entry in the cache, the intruder will be finally able to mount the definitive attack which is usually directed to a trusting server.
A thorough discussion on ARP spoofing (and on IP spoofing in general) can be found in [16].

Preventing an ARP Spoof:
As a basic precaution, trusting machines should load the hardware address of trusted machines as **permanent** entries in their ARP cache. Permanent entries do not expire after a few minutes, and can be manually inserted using the command *arp* (UNIX and Windows 95/NT).
Alternatively, a secure ARP server should be considered. An ARP server responds to ARP requests on behalf of another machine by consulting permanent entries in its own ARP cache. Even safer is to have trusting machines configured to use ARP replies coming from the ARP server rather than replies from other sources (usually a difficult task, though). Finally, since the use of routers removes the threat of ARP spoofing between IP subnets, the separation of trusted hosts from vulnerable subnets should be considered.

Detecting an ARP Spoof:
A host may attempt to detect illegitimate use of its IP address by checking, for every ARP request received, if the sender IP address matches its own. Besides, hosts can be arranged to send out an ARP request for their own IP address both on system startup and periodically thereafter. Eventual ARP replies would indicate an ARP spoof.
A server may also attempt to detect an ARP spoof by one of its clients. This can be done by querying ***RARP*** (Reverse Address Resolution Protocol) ***servers*** to cross-check the IP-to-hardware address association contained in each ARP reply received. RARP servers, maintain a database of hardware addresses and the associated IP addresses.

5.1.2. Routers and Route Spoofing.

On the Internet, both hosts and routers constantly take part in routing decisions. The destination IP address of every datagram (or fragment) arriving to a machine's network layer, is checked to decide whether the datagram should be routed to an interface in the same (sub)network (including direct deliveries to the machine itself) or forwarded to the next-hop router.
Attacks aiming to forge or tamper with routing tables are the basis of ***Route Spoofing***. Route spoofing can be achieved in several ways, all of which involve getting Internet machines to misdirect non-locally delivered IP datagrams.

ICMP-Based Route Spoofing:

An ***ICMP redirect*** error message is sent by a router to the sender of an IP datagram when the datagram should have been sent to a different router [17]. The datagram itself does not need to be re-sent because the router sending the ICMP redirect has already forwarded the datagram to the right router. As a machine receiving an ICMP redirect message typically updates its routing table, route spoofing can be achieved just by sending illegitimate redirect messages.
Note that even if a machine ignores redirect messages its datagrams are still delivered (not so efficiently, though). So, ICMP redirect spoofing can be avoided by configuring hosts to ignore redirect messages. Besides, for every redirect message received, a check (using the **permanent** entries in the ARP cache) should be made to verify that the message is from a router currently used by the machine.

RIP-Based Route Spoofing:
Modern computer networks generally use dynamic

routing algorithms. A very popular dynamic algorithm is ***distance vector routing*** **[18]**. Distance vector routing operates by having each router maintain a table (i.e., a vector) giving the best known distance to each destination and which line to use to get there. The metric used might be number of hops, time delay, total number of packets queued, or something similar. ***RIP*** **[19]** is a widely used implementation of vector-distance routing. It partitions participants into *active* and *passive* (silent) machines. A router running RIP in active mode broadcast, every 30 seconds, a message consisting of pairs, where each pair contains an IP **network** address and an integer distance to that network (measured in hops). These advertisements are used by receiving neighbors (routers and hosts) to update their routing table. Passive machines just listen and update their tables, they do not advertise. Only routers can run RIP in active mode; hosts must use passive mode.
One simple way to route spoof, is to broadcast illegitimate route information via UDP on port 520 (RIP's well-known port). This can be done from almost any PC by users with special privileges to use RIP. All passive participants will be affected. Besides, if the set of passive participants includes one or more routers then the damage can be widespread.
RIP-based route spoofing can be prevented either by disallowing routers to use RIP passively or by allowing them only a limited passive use of RIP.

Source Routing-Based Attacks:
Normally IP routing is dynamic with each router making a decision about which next-hop router to send the datagram to. However, IP can optionally use ***source routing***. The idea behind source routing is that the sender specifies the route. Two forms are provided:
- *Strict* source routing: The *exact* path is specified.
- *Loose* source routing: The sender specifies a list of IP address that the datagram must traverse, but the datagram can also pass through intermediate routers.

Besides, the *Host Requirements* RFC specifies that a TCP client must be able to specify a source route, and that a TCP server must be able to receive a source route, and use the reverse route for all segments on that TCP connection (if a newer source route is received, the earlier one is overridden).
IP spoofing attacks based on source routing, are usually mounted with the aid of route spoofing. The following example illustrates how an attack of this type is carried on:

The server **target.com** extends trust to several hosts including **friend.com** (whose legitimate IP address is vvv.xxx.yyy.zzz). An attacker operating the host **evil.com** wants to impersonate **friend.com** to obtain some services from **target.com**. First of all, the first-hop router from **evil.com** is set up by the attacker (preparing the route spoof) to route to **evil.com**'s network any arriving datagram containing vvv.xxx.yyy.zzz as the destination address. Then the IP address vvv.xxx.yyy.zzz is illegitimately assigned to **evil.com** by the attacker. Finally, when **evil.com** begins to send (IP-spoofed) packets to **target.com**, *source routing* (with the first-hop address in the route) is used (otherwise answers would be properly forwarded to **friend.com**). So, when the server answers (using the reverse route to vvv.xxx.yyy.zzz), the packets are actually going to the compromised first-hop router and getting routed to **evil.com**.
This type of attack can be prevented, of course, by disallowing source routing on **target.com**'s network.

5.1.3. DNS Spoofing.

When *resolver* software on a host needs to convert a *domain name* (e.g., jupiter.cs.yl.edu) to an IP address (e.g., 127.0.0.127) it sends an address lookup query to a DNS name server. Similarly, a *reverse* lookup query is sent when an IP address is to be converted to a domain name.
DNS Spoofing may occur whenever a DNS server gets compromised by a security attack that forge or tamper with its tables. As the responses from a DNS server are trusted by all hosts on the Internet, a compromised DNS server can direct clients to connect to illegitimate servers, or deceive servers trying to verify if an IP address corresponds to the name of a trusted client.
DNS spoofing can be partially prevented by maintaining a local database of domain names and the associated IP addresses (e.g., UNIX's /etc/hosts file). Every server's database should contain, at least, the associations corresponding to the server's trusted hosts.
Besides, an attacker trying to impersonate a trusted client, usually modifies the reverse lookup tables maintained by the DNS server that is *authoritative* (i.e., directly responsible) for the records related to the attacker machine's IP address. However, the correct association between the domain name of the legitimate client and its IP address is maintained by the DNS server that is authoritative for the client's domain name. Not only the two authoritative servers need not to be physically the same one, but even if they are, the tables for reverse and forward lookups are maintained on separate files.
So, even if the reverse lookup table gets compromised, chances are good that the forward lookup table remains sound (particularly if the attack comes from an external network). Therefore as a defense to DNS spoofing, all responses to reverse lookup queries should be

cross-checked by making a forward lookup query to detect possible inconsistencies.
The following example will illustrate how DNS spoofing is used to mount attacks:

An attacker running **evil.com** finds r* programs on **target.com** and modifies the reverse DNS entries for **evil.com** (on **compromised.dns.com**) to look like **friend.com** (a host trusted by **target.com**). When **evil.com** connects to **target.com**, the latter sends to **compromised.dns.com** a reverse lookup query (using the IP address received from **evil.com**) and gets back the (spoofed) name of **friend.com**.
As explained before, the use of products that cross-check the responses to reverse lookup queries (e.g., **TCP Wrapper [20]**) may help to prevent this type of attack.

5.1.4. TCP Connection Spoofing Blind Attacks.

A ***TCP connection spoofing attack*** is a very complex (*IP spoofing*-based) 'blind' attack. The following scenario outlines a typical TCP connection spoofing attack:
The Hosts: **Target** is a server trusting the host **Friend**, **Evil** is the attacker's machine and **Unreach** is an unreachable host.
The Events: **Evil** (impersonating **Friend**) starts a TCP connection with **Target** which, in turn, sends a reply (related to the connection establishment protocol) to the real **Friend** because neither DNS spoofing nor source routing is being used.
At this point, **Evil** is facing two problems:
1) It doesn't know what the answer from **Target** was, so it cannot be sure about the exact contents of the message that must be sent to **Target** to continue with the connection establishment protocol.
2) It must block the deliver (to **Friend**) of messages being sent by **Target**, otherwise these unexpected messages would induce **Friend** to ask **Target** to abort the connection being established (which would frustrate the attack).
Since the attacker has been gathering enough statistics as to predict what should be answered to **Target** to continue with the connection establishment, a new message containing a guessed reply is sent from **Evil** (impersonating **Friend**) to **Target**. If the attacker is correct in her prediction, the connection is established, and **Target** is compromised. Generally, after compromise, the attacker will use r* commands to insert a *backdoor*.
Meanwhile, to deal with the second problem, **Evil** (this time impersonating **Unreach**) has been flooding **Friend** with TCP connection requirements directed to the TCP port it desires disabled. Since the reply messages that are being sent from **Friend** to **Unreach** will never be acknowledged, **Friend**'s queue of incomplete connections will keep increasing until a limit is reached after which all incoming packets related to connection establishment are silently discarded by TCP (including the ones coming from **Target**).

Now, to better understand the above scenario, some related details will be explained:

• *What is attacked?*: The attack is usually directed either to port TCP 513 (*rlogin*) or to port TCP 514 (*rsh*). Often, the command 'echo "+ +" >> ~/.rhosts' (used in UNIX to extend trust to *any* user from *any* host) is executed to install a backdoor.

• *Why is it called a 'blind' attack?*: Because **Evil**'s TCP software never 'sees' any message from **Target**'s TCP (which is sending to **Friend** all the datagrams related to the fake connection). Hence **Evil** must rely exclusively on guessing.

• *What is guessed?*: TCP is a connection-oriented, reliable transport protocol. Connection-oriented means the two applications using TCP must establish a TCP connection with each other before they can exchange data. Reliability is provided in TCP by the use of checksums, timers, data sequencing and acknowledgments. By assigning a sequence number to **every** byte transferred, and requiring an acknowledgment from the other end upon receipt, TCP can guarantee reliable delivery. Sequence numbers are used to ensure proper ordering of the data and to eliminate duplicate data bytes. Note that in a TCP session there are usually two streams of data (every end point is receiving from one of them and sending through the other). So an ***ISN*** (initial sequence number) must be assigned to each stream when the connection is being established. To see how it is done, let's suppose that **C** is a client wishing to connect to the server **S**, and analyze the connection establishment process (often called the *three-way handshake*):

1	**C**	---SYN XX--->	**S**
2	**C**	<---SYN YY/ ACK XX+1---	**S**
3	**C**	---ACK YY+1--->	**S**

1. **C** sends a TCP message (known as SYN request) to **S** with the special flag SYN (SYNchronize sequence numbers) set to ON. A SYN request specifies the port number of the server that the client wants to connect to, and the client's ISN (XX in this example).

2. The server responds with its own SYN message containing **S**'s ISN (YY) and acknowledging **C**'s SYN by specifying that the **next** byte expected from **C** is the byte numbered XX+1.
3. **C** acknowledges the SYN message from **S**, and data transfer may take place.

The problem with the blind attack outlined in the scenario is that **Evil** never sees the second message, which is in fact sent to **Friend**. So what is to be guessed by the attacker is YY (in order to have Evil sending the ACK YY+1 message). However, this is not an easy task because each TCP maintains a 32-bit ISN counter that is incremented by 64,000 every half-second, and, additionally, by 64,000 each time a connection is established.
To get an idea of where in the 32-bit sequence number space **Target**'s TCP is, the attacker establishes several connections to a TCP port on **Target**, and stores the final ISN received. Besides, the attacker calculates the average RTT (round-trip time) from **Evil** to **Target** to **Evil** (most likely by using ICMP Ping messages). Now the attacker has the baseline (the last received ISN) and a good idea of how long it will take an IP datagram to travel across the Internet to reach **Target** (approximately half the average RTT, as most times the routes are symmetrical). So, the attacker immediately proceeds to initiate the three-way handshake (each interim connection would increment the ISN by 64,000) and finally the spoofed segment with the predicted ACK is sent to **Target**. As said before, if the guess is correct, **Target** is compromised.

• *What must be disabled on **Friend**?*: In the SYN request sent by **Evil** (impersonating **Friend**) to **Target**, not only the destination port number is specified, but also the source port number is included (as in every TCP segment). So when **Target** answers by sending to **Fiend** the second segment in the three-way handshake protocol (SYN/ACK) it will use as destination port number, the same one he received as source port number in the SYN request. Therefore, what must be disabled on **Friend** is precisely this port. In the outlined scenario a ***Denial of Service*** attack known as ***TCP SYN flooding*** attack (or just SYN attack) is used to disable the port. Denial of Service attacks (and SYN attacks) are discussed in section **5.3.** (**5.3.3.**).
• *How can this attack be prevented?*: This attack can be prevented by disabling all the r* commands, or (if trust is extended only to local hosts) by having the router(s) deny any packet coming from outside with a source IP address corresponding to a local host. RARP queries could be used to detect attacks coming from machines on the same physical network as the target server.

An exhaustive discussion on TCP connection spoofing attacks can be found in **[21]**.

5.2. Hijacking.

In section **5.1.4.** a basic, though complex, form of connection spoofing was analyzed. The possibility of an attack of this type, was first discussed by R. Morris **[22]**. Morris' attack can easily be extended by sniffing the network somewhere between the (trusted) client and the (target) server. The Sniffer could be used to get the server's ISN, and, even better, to establish a full duplex TCP connection with the server (assuming the real trusted client is down, or under a *denial of service* attack).
Note however that Morris' attack relies on the trusted hosts identification scheme, and is thus useless in situations where password authentication is required.
A different, yet related, type of attack based on sniffing and IP spoofing is known as ***Hijacking*** or ***TCP session hijacking***.
TCP hijacking attacks are mounted to take over existing connections. Intruders bypass one-time passwords and other strong authentication schemes by hijacking the connection after the authentication is complete. Suppose that a legitimate user connects to a remote site through a login or terminal session, if the intruder hijacks the connection after the user completed the authentication, the remote site is compromised. An even worse situation occurs if the user has logged into the remote system as ***root***, then the attacker might issue a command to change the superuser password, or add a privileged account to the password database.
Sniffers and *hijacking software* are the basic tools used to mount hijacking attacks.
A truly skillful hijack can be fully accomplished without divulging a single clue to the user, weaker implementations (which can be used to hijack idle terminal sessions) echo to the user the attacker's keystrokes and responses from the remote host.
An interesting hijacking attack is described in **[23]**. The proposed attack is based on the creation of a desynchronized state (with mismatching sequence numbers) on each end of a TCP connection so that the two points cannot exchange data any longer. A third party host (the intruder's host) is then used to create acceptable packets (mimicking the real ones) for both ends. Although some flaws related to the attack can be used to detect it, chances are good that it can be mounted without detection.
The best way to reduce the threat of a TCP connection spoofing is to use an encryption-based terminal protocol. These protocols can limit the consequences of introducing fake data on the connection, because even if the receiver accepts the data as valid, the command

interpreter will not be able to make sense of it.
More details on hijacking can be found in [24].

5.3. Denial of Service.

In a ***denial of service*** attack (*D.O.S. attack*), one user takes up so much of a shared resource that none of the resource is left for other users [25]. There are two types of denial of service attacks. The first type of attack attempts to damage or destroy resources so nobody can use them. The second type of attack overloads some system service or exhausts some resource, thus preventing others from using the service.
Networks are vulnerable to several DOS attacks. Three important types will be discussed in the following sections: *Service Overloading*, *Message Flooding* and *Clogging*.

5.3.1. Service Overloading.

Service overloading occurs when floods of requests are made to a server process on a single computer. Usually these *brute force* attacks can cause the system to be so busy that it is unable to process regular tasks in a timely fashion. Many requests will be discarded and, in some extreme cases, the attacked host will crash. A special case of service overloading results when a server process is forced to consume so many resources as to cause its host to crash. A typical example of this type of attack is known as ***Finger Bomb***. *Finger* is a client, used to gather information about users of a host. If John Doe has an account on the host **victim.com**, any user on **snoop.com** could 'finger' John Doe's account by executing the command:
snoop> finger johndoe@victim.com.
This finger of course is recognized as coming from snoop.com. However if the following command were used:
snoop> finger johndoe@victim.com@third.com
it would effectively appear as johndoe being fingered at victim.com from **third.com**. Since finger servers accept nested queries, an attacker could execute:
snoop> finger johndoe@@@@@...@@victim.com
This would cause victim.com to finger itself recursively. If victim.com fingers itself recursively enough times, then memory, swap space, and hard drive space will eventually fill up, causing the machine to crash.
An obvious, though drastic, way to prevent this attack is disabling the finger service.

5.3.2. Message Flooding.

Message flooding occurs when a user slows down a system on the network to prevent the system from processing its normal workload. Message flooding attacks are frequently directed against authentication servers. Under a message flooding attack, an authentication server can become so loaded as to be unable to respond requests coming from its clients. This will allow the attacker's machine to easily masquerade as the legitimate authentication server and, by answering with bogus information to authentication queries, to log into privileged accounts.
Often, message flooding is accomplished just by bombarding a server with thousand of ICMP *echo request* messages (using a *ping* program).
Message flooding can also be used to congest network traffic. A simple way to do this is by piping packets between two ***well-known ports*** (ports used to provide standard TCP/UDP services) to create an infinite sequence (e.g., sending *chargen* packets [port 19] over to the *echo* port [port 7]). Of course, this specific type of message flooding can be easily prevented by turning off services that might be used to cause infinite sequences (there are also some filtering programs that prevent redirecting data between selected ports).

5.3.3. Clogging.

The implementation of the three-way handshake protocol used by TCP to establish connections (see **5.1.4.**), can be abused to mount a very nasty denial of service attack known as ***TCP SYN flooding*** attack (or just SYN attack). The potential for abuse arises at the point where the server system has sent an acknowledgment (SYN/ACK) back to client but has not yet received the ACK message. This is what is known as a half-open connection. The server has built in its system memory a queue containing all pending connections. This queue is of finite size, and it can be made to overflow by intentionally creating too many partially open connections. This can be accomplished by flooding a server port with spoofed SYN messages. As the packets are spoofed to impersonate unreachable clients, the protocol will never be completed, and thus the half-open queue will eventually fill causing the server to be unable to accept any new incoming connections.
Although the half-open connections will eventually expire, the attacking system can just keep sending spoofed SYN requirements faster than the victim system can expire the pending connections.
TCP SYN flooding has been used to mount major attacks to Internet Service Providers. The services themselves are not harmed, just the ability to provide them is impaired.
Blocking an ongoing attack by having a router deny any packet coming from a specific IP address is not a solution because usually each of the source addresses

in the spoofed packets is randomly chosen, by the attacker's software, from an array of unreachable IP addresses.
Several patches have been released to make UNIX kernels SYN-attack resistant. If a patch is not available, then, at least, the number of half-open connections allowed should be increased and the amount of time that a connection is allowed to stay in a half-open state should be reduced.
Additional details on TCP SYN flooding can be found in [26].

References.

[1] A. Silberschatz, P. Galvin, **Operating Systems Concepts**. Addison-Wesley, Fourth Edition, **1994**, pp. 431.

[2] C.H. Bennett, G. Brassard, A.K. Ekert, **Quantum Cryptography**. Scientific American, v. 267, n.4, **1992**, pp. 50-57.

[3] Bruce Schneier, **Applied Cryptography**. Wiley, Second Edition, **1996**, pp. 554-557.

[4] W. Stallings, **Network and Internetwork Security**. Prentice Hall, **1995**, pp. 214-217.

[5] W. Stevens, **TCP/IP Illustrated, Vol. 1**. Addison-Wesley, **1994**.

[6] D. Comer, **Internetworking with TCP/IP Vol. 1**. Prentice Hall, Third Edition, **1995**.

[7] A.S. Tanenbaum, **Computer Networks**. Prentice Hall, Third Edition, **1996**.

[8] **ISS Sniffer FAQ**. http://www.iss.net/.

[9] D. Atkins *et al.*, **Internet Security**. New Riders, **1996**, pp. 258-279.

[10] S. Garfinkel, **PGP: Pretty Good Privacy**. O Reilly, **1994**.

[11] **SSL Version 3.0**. http://home.netscape.com/eng/ssl3/ssl-toc.html.

[12] N. Haller, **The S/KEY One-Time Password System**. RFC 1760, Bellcore, February **1995**.

[13] J. Kohl, C. Neuman, **The Kerberos Network Authentication Service (V5)**. RFC 1510, September **1993**.

[14] **Project Loki**. Phrack Magazine, **Volume Seven, Issue Forty-Nine, File 06**. Whitepaper by route@infonexus.com for Phrack Magazine. Guild Productions, August **1996**.

[15] D.C. Plummer, **An Ethernet Address Resolution Protocol**. RFC 826, **1982**.

[16] D. Atkins *et al.*, **Internet Security**. New Riders, **1996**, pp. 257-316.

[17] W. Stevens, **TCP/IP Illustrated, Vol. 1**. Addison-Wesley, **1994**, pp. 119-123.

[18] A.S. Tanenbaum, **Computer Networks**. Prentice Hall, 3rd. ed., **1996**, pp. 355-359.

[19] C. Hedrick, **Routing Information Protocol**. RFC 1058, **1988**.

[20] K. Siyan, C. Hare, **Internet Firewalls and Network Security**. New Riders, **1995**, pp. 304-306.

[21] **IP-spoofing Demystified**. By route@infonexus.com Guild Productions, **June 1996**.

[22] R. Morris, **A Weakness in the 4.2 BSD UNIX TCP/IP Software**. Computing Science Technical Report 117. AT&T Bell Laboratories, **1985**.

[23] L. Joncheray, **A Simple Active Attack Against TCP**. Merit Network, Inc. lpj@merit.edu, **April 1995**.

[24] **IP Spoofing Attacks and Hijacked Terminal Connections**. CERT Advisory CA-95:01. Available at http://www.cert.org and at ftp://info.cert.org/pub/cert_advisories/ **1995**.

[25] S. Garfinkel, G. Spafford, **Practical UNIX & Internet Security**. O Reilly, 2nd Edition, **1996**, pp. 759-778.

[26] **TCP SYN Flooding and IP Spoofing Attacks**. CERT Advisory CA-96.21. Available at http://www.cert.org and at ftp://info.cert.org/pub/cert_advisories/ **1996**.