Chapter 5

Network Security I

Contents

5.1 Network Security Concepts

The Internet was originally conceived, during the Cold War, as a way of creating a communication network that was robust enough to survive a military attack. For this reason, rather than basing communication on switched paths that connect communicating parties, the Internet was designed so that communication occurs through sequences of data packets. A data ***packet*** is a finite-length set of bits, which is divided into two parts: a ***header***, which specifies where the packet is going and contains various overhead and bookkeeping details, and a ***payload***, which is the actual information that is being communicated. So if two entities wish to communicate using the Internet, they must chop their messages into packets, attach a header on the front of each one, and then have those packets find their way through the Internet to reach their respective destinations. In this chapter, we explore the underlying technologies that make the Internet possible, including its security risks and some defense mechanisms.

5.1.1 Network Topology

A network's connection structure is known as its ***network topology***. The computers in a network are ***host nodes*** that can be sources and destinations of messages, and the routers in the network are ***communication nodes*** through which messages flow. (See Figure 5.1.) The physical connections between nodes define the channels through which messages travel, so that packets move by being passed from one node to the next in order to get from their source node to their destination node.

A private network composed of computers in relatively close proximity to each other is known as a ***local area network***, or ***LAN***. In contrast, the Internet is what is referred to today as a ***wide area network***, or ***WAN***, composed of many machines and smaller networks spread out over great distances. In addition, the routers in wide-area networks on the Internet are partitioned into clusters, which are called ***autonomous systems*** (ASs). Each autonomous system is controlled by a single organizational entity, which determines how packets will be routed among the nodes in that AS. Typically, this routing within an AS is done using shortest paths, so that the number of hops to route a packet from one node to another in this AS is minimized and routing cycles are avoided. The routing between multiple ASs, on the other hand, is determined by contractual agreements, but it is still designed to avoid loops.

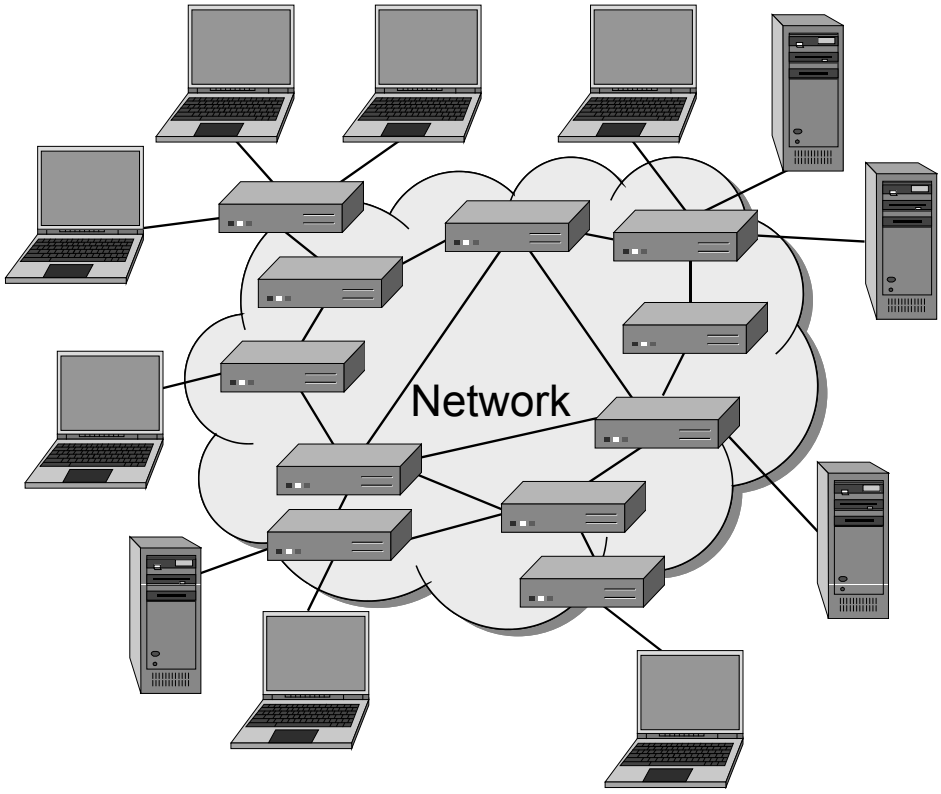

Figure 5.1: A computer network composed of host nodes (shown as computers on the periphery) and communication nodes (shown as routers in the interior).

5.1.2 Internet Protocol Layers

Before delving into the wide range of security issues the Internet creates, it is important to understand the underlying building blocks that comprise it. The architecture of the Internet is modeled conceptually as being partitioned into layers, which collectively are called the ***Internet protocol stack***. Each layer provides a set of services and functionality guarantees for higher layers and, to the extent possible, each layer does not depend on details or services from higher levels. Likewise, the interface each layer provides to higher levels is designed to provide only the essential information from this layer that is needed by the higher levels—lower-level details are hidden from the higher levels. The exact number and names of the layers of the Internet protocols vary slightly, but is usually five or seven, depending on what source we consider as authoritative.

Five Conceptual Layers for Internet Communication

The following division into five layers is fairly standard.

1. ***Physical layer***. The task of the physical layer is to move the actual bits between the nodes of the network, on a best effort basis. For example, this level deals with details related to whether connections are done with copper wires, coaxial cables, optical-fiber cables, or wireless radio. The abstraction it provides to the next higher level is an ability to transmit bits between a pair of network nodes.

2. ***Link layer***. The task of the link layer is to transfer data between a pair of network nodes or between nodes in a local-area network and to detect errors that occur at the physical layer. This layer, for instance, deals with the logical aspects of sending information across network links and how to find good routing paths in a local-area network. It includes such protocols as Ethernet, which is used to route packets between computers sharing a common connection. The link layer provides a grouping of bits into ordered records, called ***frames***. The link layer uses 48-bit addresses, called ***media access control addresses*** (***MAC addresses***).

3. ***Network layer***. The task of the network layer, which is also known as the ***Internet layer*** for the Internet, is to provide for the moving of packets between any two hosts, on a ***best effort*** basis. It provides a way of individually addressing each host using a numerical label, called its ***IP address***. The main protocol provided by this layer is the ***Internet Protocol*** (***IP***), which is subdivided into a version 4 (IPv4), which uses 32-bit IP addresses, and a version 6 (IPv6), which uses 128-bit IP addresses. Best effort basis means there are no guarantees that any given packet will be delivered. Thus, if reliable delivery is required by an application, it will have to be provided by a higher layer.

4. ***Transport layer***. The task of the transport layer is to support communication and connections between applications, based on IP addresses and ***ports***, which are 16-bit addresses for application-level protocols to use. The transport layer provides a protocol, the ***Transmission Control Protocol*** (***TCP***), which establishes a virtual connection between a client and server and guarantees delivery of all packets in an ordered fashion, and a protocol, the ***User Datagram Protocol*** (***UDP***), which assumes no prior setup and delivers packets as quickly as possible but with no delivery guarantees.

5. ***Application layer***. The task of the application layer is to provide protocols that support useful functions on the Internet, based on the services provided by the transport layer. Examples include HTTP, which uses TCP and supports web browsing, DNS, which uses UDP and supports the use of useful names for hosts instead of IP addresses, SMTP and IMAP, which use TCP and support electronic mail, SSL, which uses TCP and supports secure encrypted connections, and VoIP, which uses UDP and supports Internet telephone messaging.

The Open Systems Interconnection (OSI) model differs slightly from that above, in that it has seven layers, as the application layer is divided into a strict application layer, for host application-to-network processes, a presentation layer, for data representation, and session layer, for interhost communication. We will use the five-layer model in this book, however, which is called the TCP/IP model, so as to focus on the security issues of the Internet. A packet for a given layer in this model consists of the data to be transmitted plus metadata providing routing and control information. The metadata is stored in the initial portion of the packet, called ***header*** and sometimes also in the final portion of the packet, called ***footer***. The data portion of the packet is referred to as the ***payload***. For all but the topmost layer, the payload stores a packet of the layer immediately above. This nesting of packets is called ***encapsulation*** and is illustrated in Figure 5.2.

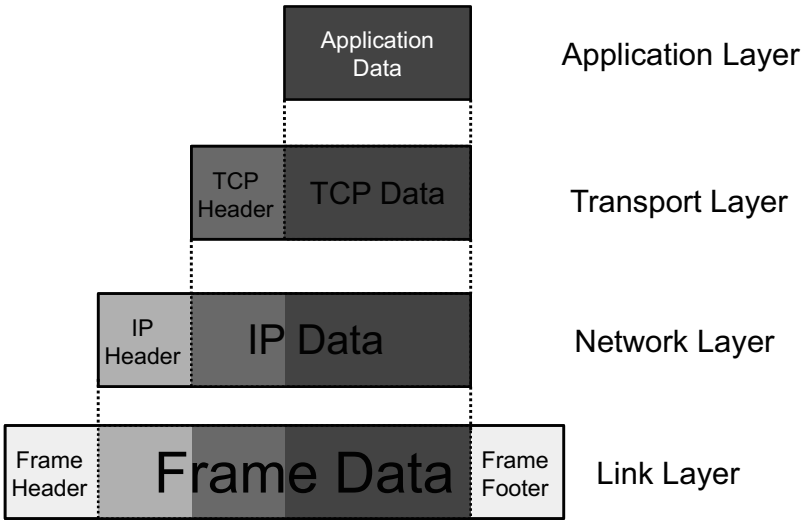

Figure 5.2: Packet encapsulation in the link, network, transport, and application layers of the Internet protocol stack. Each packet from a higher layer becomes the data for the next lower-layer packet, with headers added to the beginning, and, for frames, a footer added at the end.

Using the Internet Protocol Suite

The Internet Protocol stack provides a useful set of functions and abstractions that make the Internet possible, but we should point out that these functions and abstractions were, for the most part, designed during a time when the Internet was almost exclusively populated by people with no malicious intent. A challenge for today, then, is to figure out ways of building in security and safeguards into Internet protocols, which is the main theme of the remainder of this chapter and the next.

The layered model used for the Internet Protocol Suite helps system designers to build software that uses appropriate services and provides the right service guarantees, without troubling with unnecessary implementation details. For example, a web server transmitting content to a client's web browser would probably do so using the HTTP application-layer protocol. The HTTP packet would most likely be encapsulated in the payload of a TCP transport-layer packet. In turn, the TCP packet would be contained in the payload of an IP packet, which in turn would be wrapped in an appropriate link-layer protocol such as Ethernet, to be transferred over a physical means of transmission. (See Figure 5.3.)

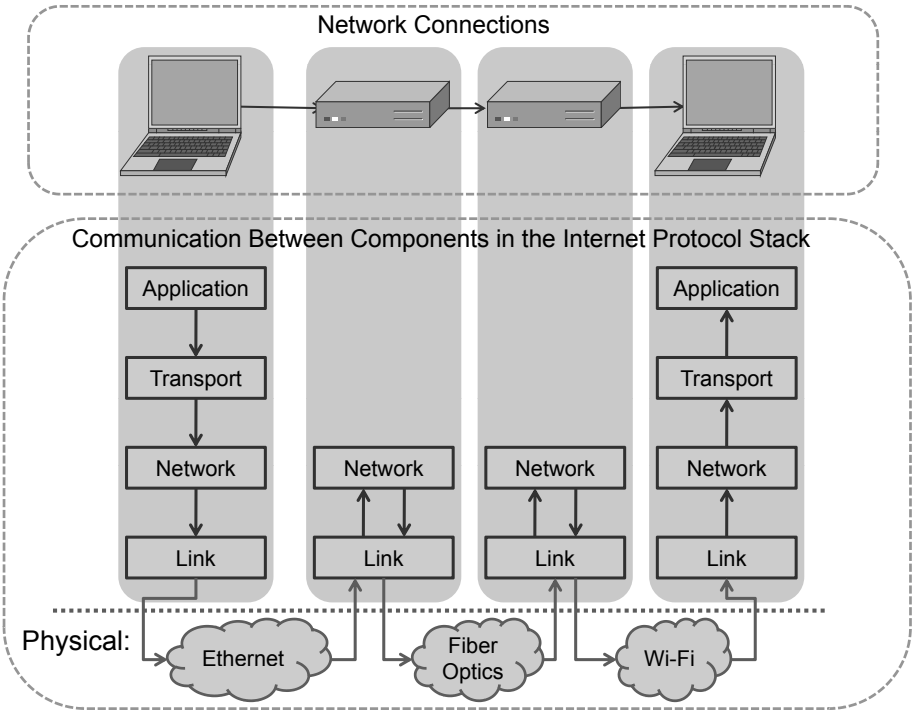

Figure 5.3: Connections and communication needed to send data from a host through two routers to another host.

5.1.3 Network Security Issues

Connecting computers to a network, like the Internet, so they can exchange data and share computations allows for huge benefits to society. Indeed, it is hard to imagine what life today would be like without the Internet. But computer networking also allows for a number of attacks on computers and information. So let us revisit some of the principles of computer security, which were discussed in Chapter 1, focusing now on how they are impacted by computer networking.

How Networking Impacts Computer Security Goals

- ***Confidentiality***. There is no requirement, in any of the layering abstractions discussed above, that the contents of network packets be kept confidential. In fact, standard protocols for each layer don't encrypt the contents of either their headers or their data. Thus, if network communications should be kept confidential, then encryption should be done explicitly. This encryption could either be done at the application layer (as in the HTTPS protocol) or by revising a lower-layer protocol to include encryption, such as in the IPsec specification (Section 6.3.2).

- ***Integrity***. The headers and footers that encapsulate data packets have, at each layer, simple checksums to validate the integrity of data and/or header contents. These checksums are effective at determining if a small number of bits have been altered, but they are not cryptographically secure, so they don't provide integrity in the computer security sense. Thus, if true integrity is required, then this should also be done at the application layer or with alternative protocols at lower layers.

- ***Availability***. The Internet was designed to tolerate failures of routers and hosts. But the sheer size of the Internet makes availability a challenge for any network object that needs to be available on a 24/7 basis. For instance, web servers can become unavailable because they become bombarded with data requests. Such requests could come from hoards of legitimate users suddenly interested in that web site or from an attack coming from many compromised hosts that is meant to create a denial of service for the web site. Thus, to achieve availability at the scale of the Internet, we need network applications that can scale with increases in communication requests and/or block attacks from illegitimate requests.

- ***Assurance***. As a default, a packet is allowed to travel between any source and destination in a network. Thus, if we want to introduce permissions and policies that control how data flows in a network, these have to be implemented as explicit additions. For instance, network firewalls are designed to block traffic in and out of a network domain if that traffic violates policies set by administrators.

- ***Authenticity***. The headers and footers for the standard Internet protocols do not have a place to put digital signatures. Indeed, in the Internet Protocol stack, there is no notion of user identities. Data is exchanged between machines and processes, not people. Thus, if we want to introduce identities and allow for signatures, then we must do so explicitly at the application layer or with an alternative protocol.

- ***Anonymity***. Since there is no default notion of identity of users of the Internet, it has a built-in anonymity. This anonymity is probably a good thing for a human rights worker reporting on abuses, but it's probably bad if it lets an identity thief steal credit card numbers without being caught. Attacks on anonymity can come from technologies that identify the computer a person is using. Likewise, people can replicate many copies of a process and place these copies at multiple hosts in the network, thus achieving a level of anonymity.

We illustrate some network attacks against these principles in Figure 5.4.

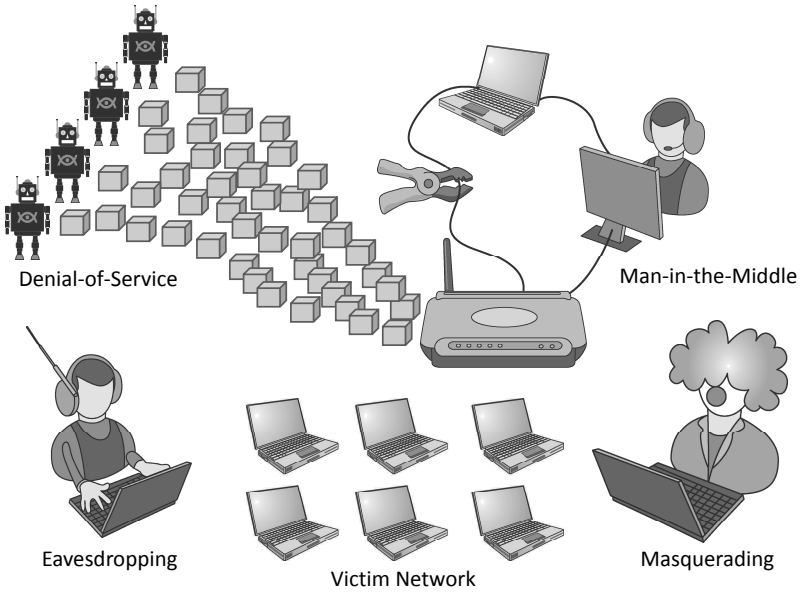

Figure 5.4: Some network-based attacks.

5.2 The Link Layer

Most modern operating systems include a TCP/IP implementation and allow programs to interact with the Internet Protocol stack via a simple interface. Operating system libraries include support for the upper levels of the TCP/IP stack, including the passing of data to physical-layer device drivers, starting with the link layer, which is right above the physical layer and provides a concept of grouping sequences of bits into frames.

5.2.1 Ethernet

One of the most popular ways to transmit Internet traffic is ***Ethernet***, which refers to both the physical medium used (typically a cable) as well as the link-layer protocol standardized as IEEE 802.3. When a frame is transmitted on an Ethernet cable, an electrical impulse is sent through that cable and received by other machines that are logically connected to that cable on the same local-area network (LAN). The portion of a local-area network that has the same logical connection is called a ***network segment***. If two machines on the same network segment each transmit a frame at the same time, a ***collision*** occurs and these frames must be discarded and retransmitted. Fortunately, the Ethernet protocol can deal with such events using a random-wait strategy. (See Figure 5.5.)

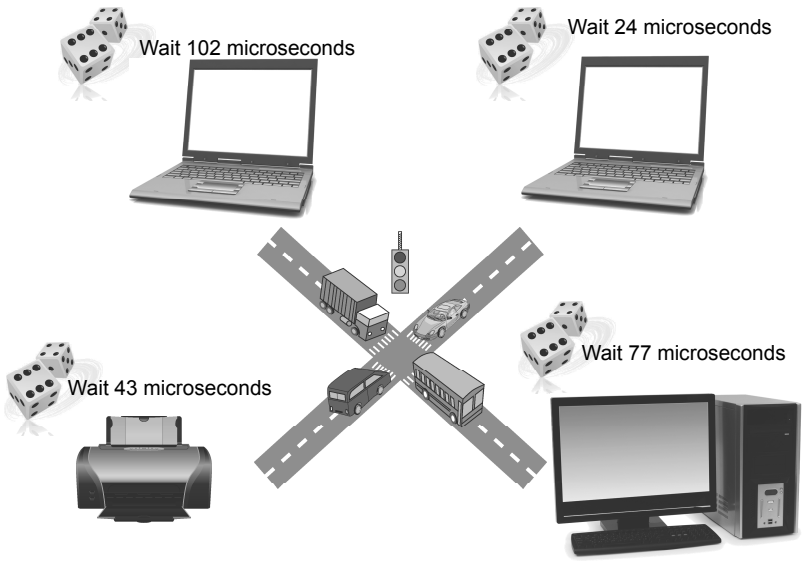

Figure 5.5: An Ethernet collision and how it is handled.

Dealing with Collisions

In the event of a collision, each of the transmitting machines waits a random amount of time, usually measured in microseconds, and then retransmits, in the hope of avoiding a second collision. If other collisions occur, then this process of randomly waiting and retransmitting is repeated. The Ethernet protocol is designed so that eventually every machine in a network segment will succeed in transmitting its frame. Incidentally, this collision resolution protocol was originally needed even for two machines connected by a single (coaxial) cable, since such cables are not bidirectional, but modern network cables can transmit data in both directions, so this collision resolution process only applies to network segments that contain more than two machines. That is, two machines connected by a modern Ethernet cable can send and receive messages without accounting for the possibility of collisions. But packet collisions can nevertheless become a major source of slowdown for local-area networks if there are larger numbers of machines logically connected to each other. Indeed, this can already be an issue with home networks, since it is not uncommon for such networks to include several computers, a couple of network printers, and at least one Wi-Fi access point. So, even for a home network, it is useful to know how to connect machines so as to minimize collisions.

Hubs and Switches

The simplest way to connect machines in a local-area network is to use an Ethernet ***hub***, which is a device that logically connects multiple devices together, allowing them to act as a single network segment. Hubs typically forward all frames to all attached devices, doing nothing to separate each attached device, much like a splitter used to double the audio signal from an MP3 player. Thus, the machines that are connected to a hub, or a set of connected hubs, form a single network segment and must all participate in the Ethernet collision resolution protocol. Hubs may generate large amounts of unnecessary traffic, since each frame is duplicated and broadcast to all the machines connected on the same network segment. In addition, the fact that all frames are forwarded to each machine in the segment, regardless of the intended destination, increases the ease of network eavesdropping, as discussed in Section 5.3.4.

Fortunately, there is a better way to connect machines in a small local-area network—namely, to use an Ethernet ***switch***. When devices are first connected to an Ethernet switch, it acts much like a hub, sending out frames to all connected machines. Over time, however, a switch learns the addresses of the machines that are connected to its various ports. Given

this address information, a switch will then only forward each frame it receives along the cable it knows is connected to the destination for that frame. Even so, if a frame is designated as one that should be broadcast to all the machines on a network segment, a switch will still act like a hub and send that frame out to all its connected machines.

The selectivity that comes from a switch learning the addresses of the machines it connects reduces the possibilities for collisions and increases the effective speed of the network, that is, its effective ***bandwidth***. In addition, a switch reduces the risks of network eavesdropping, since network frames forwarded by a switch are less likely to be seen by machines that are not destinations.

Due to decreasing costs in networking technology, switches have become the de facto standard for link layer data forwarding. We illustrate the difference between a hub and a switch in Figure 5.6.

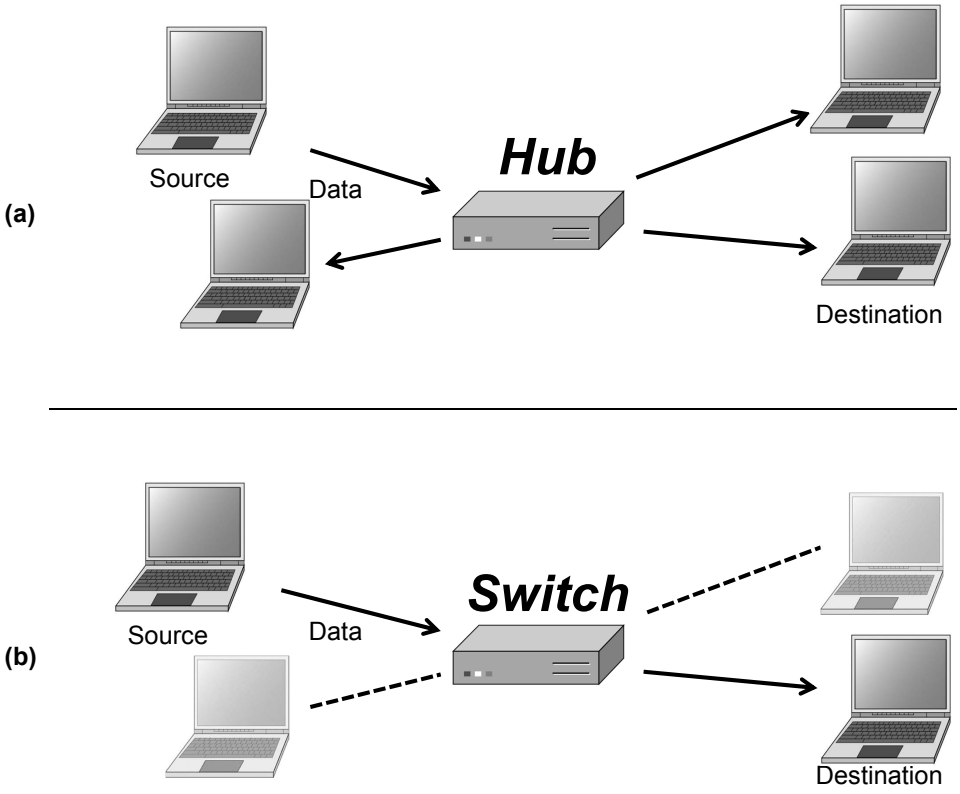

Figure 5.6: Hub vs. switch: (a) A hub copies and transmits traffic to all attached devices. (b) A switch only transmits frames to the appropriate destination device.

5.2.2 Media Access Control (MAC) Addresses

Network interfaces are typically identified by a hardware-specific identifier known as its ***media access control address*** (***MAC address***). A MAC address is a 48-bit identifier assigned to a network interface by its manufacturer. It is usually represented by a sequence of six pairs of hexadecimal digits, e.g., 00:16:B7:29:E4:7D, and every device that connects to a network has one.

MAC addresses are used in the link layer to identify the devices in a network; hence, MAC addresses are intended to be unique for each interface. Typically, the first 24 bits are a prefix identifying the organization that issued the MAC address (these prefixes are issued by IEEE). This information can sometimes be used to identify the brand or model of a particular interface on a network. Thus, the remaining 24 bits are left to a manufacturer to set so that each of its different model instances have unique MAC addresses. Fortunately, there are $2^{24} = 16,777,216$ possibilities for these 24 bits, so that even if a manufacturer has to start reusing MAC addresses, the chance of two devices on the same network having the same manufacturer-assigned MAC address is on the order of a one-in-a-million.

Despite the fact that they are designed to be unique identifiers, MAC addresses can be changed by software through the driver of the network interface. Network administrators can use this functionality to issue their own MAC addresses to network interfaces on their network. These locally administered MAC addresses are distinguished from MAC addresses issued by a manufacturer by a standardized identifier bit. In a locally administered MAC address, the second-least-significant bit of the most significant byte is set to 1, while in a manufacturer-issued MAC, this bit is set to 0. Because MAC addresses can be trivially changed using software, such as the ifconfig utility on Linux, they cannot be used as a reliable means of identifying an untrusted source of network traffic.

MAC addresses are used at the link layer to facilitate the routing of frames to the correct destination. In particular, switches learn the location of network devices from their MAC addresses and they forward frames to the appropriate segments based on this knowledge. The format of an Ethernet frame is depicted in Figure 5.7. Note that each such frame contains its source and destination MAC addresses, a CRC-32 checksum for confirming data integrity, and a payload section, which contains data from higher layers, such as the IP layer. The CRC-32 checksum is a simple function of the contents of the frame and it is designed to catch transmission errors, such as if a 0 bit in the frame is accidentally changed to a 1 during transmission. In particular, this checksum is not designed for strong authentication of device identities—it is not as secure as a digital signature.

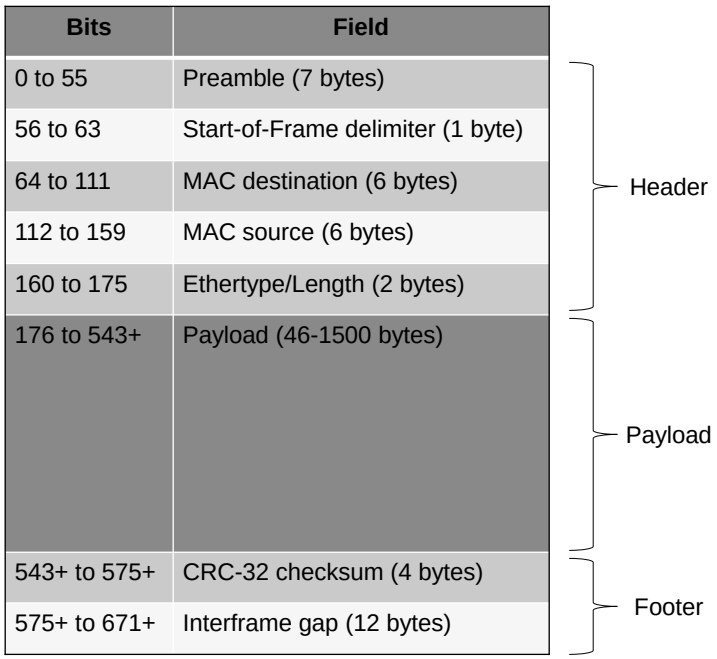

Figure 5.7: The format of an Ethernet frame.

5.2.3 ARP Spoofing

The ***Address Resolution Protocol*** (***ARP***) is a link-layer protocol that provides services to the network layer. ARP is used to find a host's hardware address given its network layer address. Most commonly, it is used to determine the MAC address associated with a given IP address, which is clearly a valuable service. Unfortunately, there is a man-in-the-middle attack against this protocol, which is called ***ARP spoofing***.

How ARP Works

Suppose a source machine wants to send a packet to a destination machine on the local-area network. At the network layer, the source machine knows the destination IP address. Since the sending of the packet is delegated to the link layer, however, the source machine needs to identify the MAC address of the destination machine. In the ARP protocol, the resolution of IP addresses into MAC addresses is accomplished by means of a broadcast message that queries all the network interfaces on a local-area network, so that the proper destination can respond.

How ARP Spoofing is Done

An ***ARP request*** for an IP address, such as 192.168.1.105, is of the type:

"Who has IP address 192.168.1.105?"

This request is sent to all the machines on the local-area network. The machine with IP address 192.168.1.105, if any, responds with an ***ARP reply*** of the type:

"192.168.1.105 is at 00:16:B7:29:E4:7D"

This ARP reply is transmitted in a frame addressed only to the machine that made the ARP request. When this machine receives the ARP reply, it stores the IP-MAC address pair locally in a table, called its ***ARP cache***, so it does not have to continually resolve that particular IP address. After this ARP resolution, the source can finally send its data to its destination.

The ARP protocol is simple and effective, but it lacks an authentication scheme. Any computer on the network could claim to have the requested IP address. In fact, any machine that receives an ARP reply, even if it was not preceded by a request, will automatically update its ARP cache with the new association. Because of this shortcoming, it is possible for malicious parties on a LAN to perform the ***ARP spoofing*** attack.

This attack is relatively straightforward. An attacker, Eve, simply sends an ARP reply to a target, who we will call Alice, who associates the IP address of the LAN gateway, who we will call Bob, with Eve's MAC address. Eve also sends an ARP reply to Bob associating Alice's IP address with Eve's MAC address. After this ***ARP cache poisoning*** has taken place, Bob thinks Alice's IP address is associated with Eve's MAC address and Alice thinks Bob's IP address is associated with Eve's MAC address. Thus, all traffic between Alice and Bob (who is the gateway to the Internet) is routed through Eve, as depicted in Figure 5.8.

Once accomplished, this establishes a ***man-in-the-middle*** scenario, where the attacker, Eve, has control over the traffic between the gateway, Bob, and the target, Alice. Eve can choose to passively observe this traffic, allowing her to sniff passwords and other sensitive information, or she can even tamper with the traffic, altering everything that goes between Alice and Bob. A simple denial-of-service attack is also possible.

The power of ARP spoofing is derived from the lack of identity verification in the Internet's underlying mechanisms. This attack requires users to take caution in securing their local networks. Fortunately, there are several means of preventing ARP spoofing, besides restricting LAN access to trusted users. One simple technique involves checking for multiple occurrences of the same MAC address on the LAN, which may be an indicator of possible ARP spoofing.

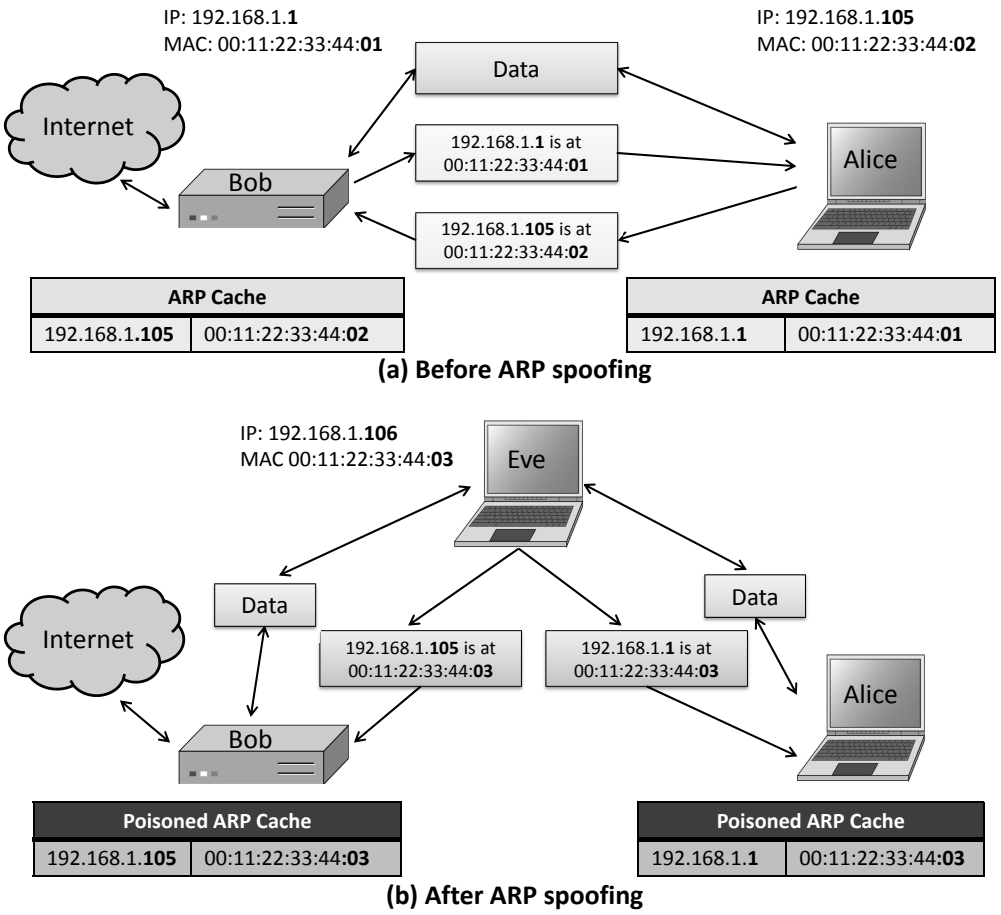

Figure 5.8: ARP spoofing enables a man-in-the-middle attack: (a) Before the ARP spoofing attack. (b) After the attack.

Another solution, known as ***static ARP tables***, requires a network administrator to manually specify a router's ARP cache to assign certain MAC addresses to specific IP addresses. When using static ARP tables, ARP requests to adjust the cache are ignored, so ARP spoofing of that router is impossible. This requires the inconvenience of having to manually add entries for each device on the network, however, and reduces flexibility when a new device joins the network, but significantly mitigates the risk of ARP cache poisoning. Moreover, this solution does not prevent an attacker from spoofing a MAC address to intercept traffic intended for another host on the network.

For more complex and flexible defense techniques, many software solutions exist that carefully inspect all ARP packets and compare their contents with stored records of ARP entries, detecting and preventing spoofing. Examples include programs such as anti-arpspoof, XArp, and Arpwatch.

5.3 The Network Layer

The task of the network layer is to move packets between any two hosts in a network, on a ***best effort*** basis. It relies on the services provided by the link layer to do this. As with the link layer, there are a number of computer security issues that are associated with the network layer.

5.3.1 IP

The ***Internet Protocol*** (***IP***) is the network-level protocol that performs a best effort to route a data packet from a source node to a destination node in the Internet. In IP, every node is given a unique numerical address, which is a 32-bit number under version 4 of the protocol (***IPv4***) and is a 128-bit number under version 6 of the protocol (***IPv6***). Both the source and destination of any transmission are specified by an IP address.

Routing IP Packets

A host such as a desktop PC, server, or smartphone, employs a simple algorithm for routing packets from that host (see Figure 5.9):

- If the packet is addressed to a machine on the same LAN as the host, then the packet is transmitted directly on the LAN, using the ARP protocol to determine the MAC address of the destination machine.

- If the packet is addressed to a machine that is not on the LAN, then the packet is transmitted to a specially designated machine on the LAN, called a ***gateway***, which will handle the next step of the routing. The ARP protocol is used to determine the MAC address of the gateway.

Thus, a host typically stores a list of the IP addresses of the machines on its LAN, or a compact description of it, and the IP address of the gateway.

Once a packet has reached a gateway node, it needs to be further routed to its final destination on the Internet. Gateways and other intermediate network nodes that handle the routing of packets on the Internet are called ***routers***. They are typically connected to two or more LANs and use internal data structures, known as ***routing tables***, to determine the next router to which a packet should be sent. Given a data packet with destination t, a routing table lets a router determine which of its neighbors it should send this packet to. This determination is based on the numerical address, t, and the routing protocol that encodes the next hop from this router to each possible destination.

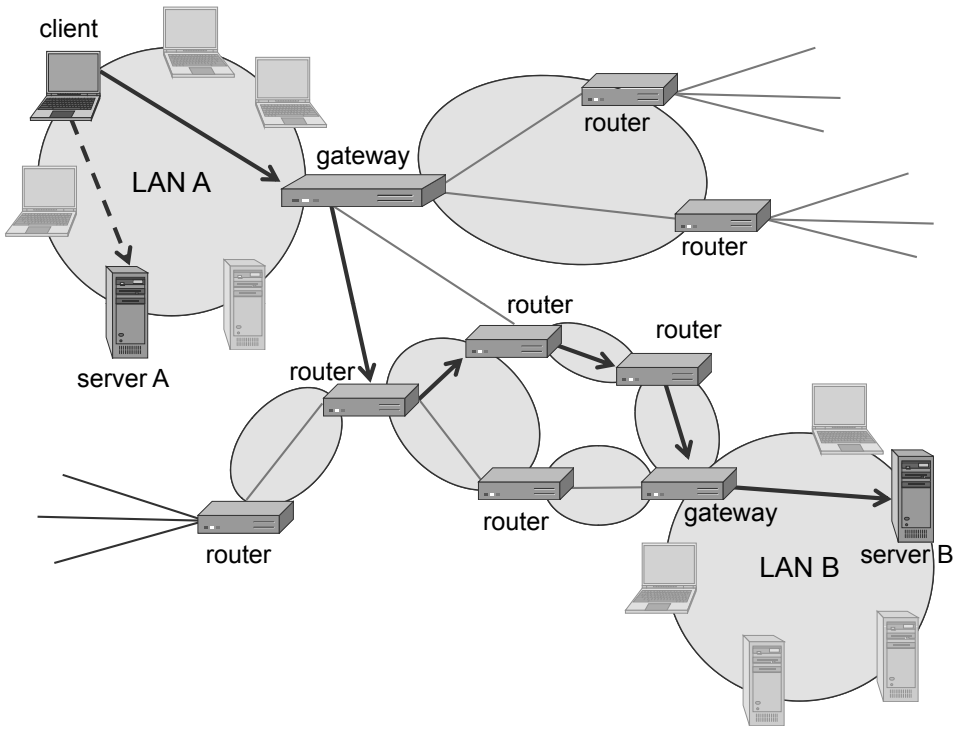

Figure 5.9: Routing on the Internet. A first packet, from the client to Server A, is sent directly over LAN A. The transmission of the first packet is shown with a dashed arrow. A second packet, from the source to Server B, is sent first to the gateway of LAN A, then forwarded by several intermediate routers, and finally delivered to Server B by the gateway of LAN B. The path followed by the second packet is shown with thick solid arrows. Adjacent routers are themselves connected via LANs. The route of a packet may not be the shortest path (in terms of number of edges or total delay) between the source and destination.

Misconfigurations in the routing tables may cause a packet to travel forever aimlessly along a cycle of routers. To prevent this possibility and other error conditions that keep unroutable packets in the network, each IP packet is given a ***time-to-live*** (***TTL***) count by its source. This TTL value, which is also known as a ***hop limit***, can be as large as 255 hops and is decremented by each router that processes the packet. If a packet's TTL ever reaches zero, then the packet is discarded and an error packet is sent back to the source. A packet with TTL equal to zero is said to be expired and should be discarded by a router that sees it.

The Structure of the Internet

Routers are designed to be very fast. For each packet received, the router performs one of three possible actions.

- ***Drop***. If the packet is expired, it is dropped.
- ***Deliver***. If the destination is a machine on one of the LANs to which the router is connected, then the packet is delivered to the destination.
- ***Forward***. If the destination of the packet does not belong to the LANs of the router, then the packet is forwarded to a neighboring router.

There are two primary protocols that determine how the next hops are encoded in Internet routing tables, ***Open Shortest Path First*** (***OSPF***) and ***Border Gateway Protocol*** (***BGP***). OSPF determines how packets are routed within an autonomous system and is based on a policy that packets should travel along shortest paths. BGP, on the other hand, determines how packets are routed between autonomous systems (ASs) and it is based on policies dictated by contractual agreements between different ASs The routes established by BGP may not be shortest paths.

Note the difference between a router and a switch. A switch is a simple device that handles forwarding of packets on a single network and uses learned associations to reduce the use of broadcasting. A router, on the other hand, is a sophisticated device that can belong to multiple networks and uses routing tables to determine how to forward packets, thereby avoiding broadcast altogether.

The bits in an IP packet have a careful structure. Each IP packet consists of a fixed-length header, which is partitioned into various fields, shown in Figure 5.10, followed by a variable-length data portion. Note that the header has specific fields, including the total length of the packet, the time-to-live (TTL) for this packet, the source IP address, and the destination IP address.

Although it does not guarantee that each packet successfully travels from its source to its destination, IP does provide a means to detect if packet headers are damaged along the way. Each IP packet comes with a checksum value, which is computed on its header contents. Any host or router wishing to confirm that this header is intact simply needs to recompute this checksum function and compare the computed checksum value to the checksum value that is stored inside the packet. Since some parts of the header, like the time-to-live, are modified with each hop, this checksum value must be checked and recomputed by each router that processes this packet. The protocol field of an IP packet specifies the higher level protocol that should receive the payload of the packet, such as ICMP, TCP, or UDP, which are described later in this chapter.

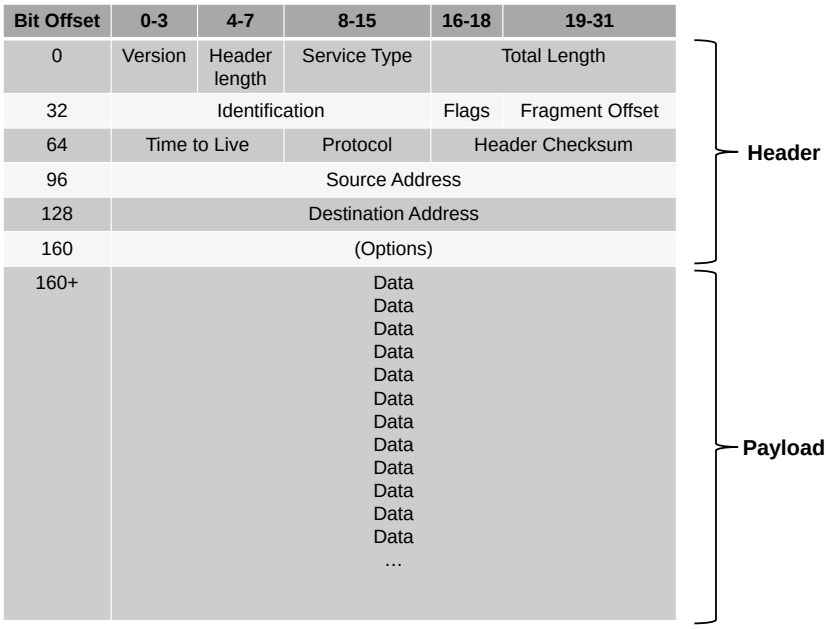

Figure 5.10: Format of an IPv4 packet.

As mentioned above, the Internet is divided into autonomous systems, so routing tables have to be able to direct traffic to clusters of nodes, not just individual destination. To facilitate this ability, the IP addressing scheme takes into account the fact that networks are partitioned into logical groupings known as ***subnetworks***, or more commonly, ***subnets***. As mentioned, IPv4 addresses are 32-bit numbers that are stored as binary but typically written as 4 bytes, such as 192.168.1.100. IP addresses can be divided into two portions, a network portion that denotes an IP prefix used by all machines on a particular network, and a host portion which identifies a particular network device. These two portions are differentiated by providing a ***subnet mask*** along with the IP address. The network portion of the IP address can be identified by bitwise ANDing the subnet mask with the IP address, and the host portion can be identified by XORing this result with the IP address. (See Table 5.1.)

		Address	**Binary**
A	IP address	192.168.1.100	11000000.10101000.00000001.01100100
B	Subnet mask	255.255.255.0	11111111.11111111.11111111.00000000
C	Network part ($A \wedge B$)	192.168.1.0	11000000.10101000.00000001.00000000
D	Host part ($A \oplus C$)	0.0.0.100	00000000.00000000.00000000.01100100

Table 5.1: Network and host portions of IP addresses and subnet masks.

Subnet masks are used to define the address range of a particular network. Ranges of IP addresses are based on the size of the organization in question. A ***Class A*** network, which is the largest, has a subnet mask of at least 8 bits and includes up to $2^{24} = 16,777,216$ unique IP addresses. Class A networks are typically reserved for large government organizations and telecommunications companies. ***Class B*** networks have at least a 16-bit subnet mask and up to $2^{16} = 65,536$ unique IP addresses; they are typically allocated for ISPs and large businesses. Finally, ***Class C*** networks have at least a 24-bit subnet mask, include up to $2^8 = 256$ unique addresses, and are assigned to smaller organizations. IP addresses with the host portion consisting of all zeros or all ones have a special meaning and are not used for to identify machines. Thus, a class C network has 254 usable IP addresses.

The original designers of the Internet could not predict the massive degree to which it would be adopted around the world. Interestingly, at the time of this writing, the total address space for IPv4 is on the verge of exhaustion: soon, all possible IPv4 addresses will be assigned. Although ***Network Address Translation***, or ***NAT*** (Section 5.4.3), delays the exhaustion of the IPv4 address space, it doesn't solve it, and an actual solution is provided by IPv6, which features 128-bit addresses.

5.3.2 Internet Control Message Protocol

The ***Internet Control Message Protocol*** (***ICMP***) is a network-layer protocol that is used by hosts to perform a number of basic testing and error notification tasks. ICMP is primarily used for network diagnostic tasks, such as determining if a host is alive and finding the path followed by a packet. ICMP packets carry various types of messages, including the following:

- ***Echo request***: Asks the destination machine to acknowledge the receipt of the packet
- ***Echo response***: Acknowledges the receipt of a packet in reply to an echo request
- ***Time exceeded***: Error notification that a packet has expired, that is, its TTL is zero
- ***Destination unreachable***: Error notification that the packet could not be delivered

Several network management tools use the above ICMP messages, including the popular ping and traceroute utilities.

Ping

Ping is another utility that uses the ICMP protocol to verify whether or not a particular host is receiving packets. Ping sends an ICMP echo request message to the destination host, which in turn replies with an ICMP echo response message. This remarkably simple protocol is often the first diagnosis tool used to test if hosts are working properly.

Traceroute

The traceroute utility uses ICMP messages to determine the path a packet takes to reach another host, either on a local network or on the Internet. It accomplishes this task with a clever use of the time-to-live (TTL) field in the IP header. First, it attempts to send a packet to the target with a TTL of 1. On receiving a packet with a TTL of 1, an intermediate router discards the packet and replies to the sender with an ICMP time exceeded message, revealing the first machine along the path to the target. Next, traceroute sends a packet with a TTL of 2. On reaching the first router in the path, the TTL is decremented by one and forwarded to the next router, which in turn sends an ICMP packet to the original sender. By incrementing the TTL field in this way, traceroute can determine each host along the path to the target. The traceroute utility is illustrated in Figure 5.11.

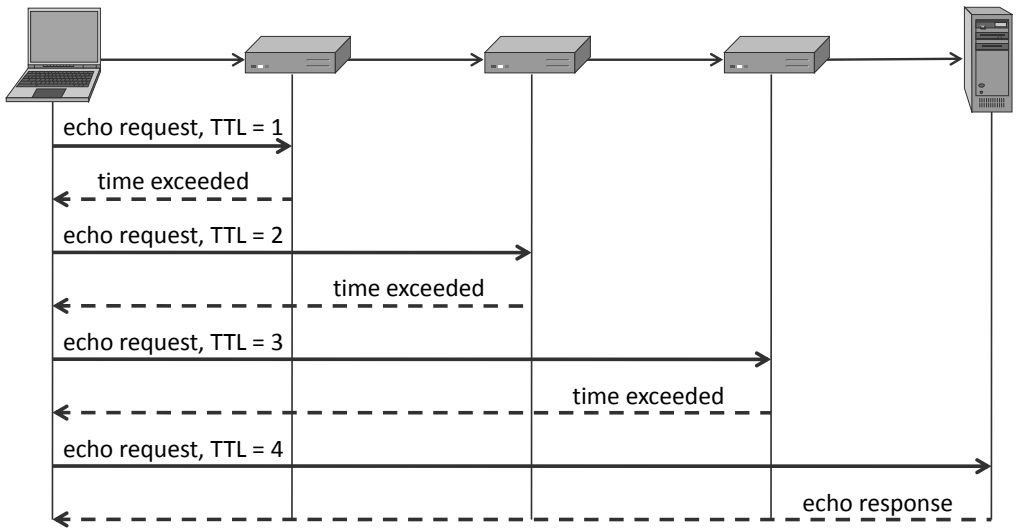

Figure 5.11: The traceroute utility.

5.3.3 IP Spoofing

Each IP packet includes a place to specify the IP addresses of the destination and source nodes of the packet. The validity of the source address is never checked, however, and it is trivial for anyone to specify a source address that is different from their actual IP address. In fact, nearly every operating system provides an interface by which it can make network connections with arbitrary IP header information, so spoofing an IP address is a simple matter of specifying the desired IP in the source field of an IP packet data structure before transmitting that data to the network. Such modification of the source address to something other than the sender's IP address is called ***IP spoofing***. (See Figure 5.12.) IP spoofing does not actually allow an attacker to assume a new IP address by simply changing packet headers, however, because his actual IP address stays the same.

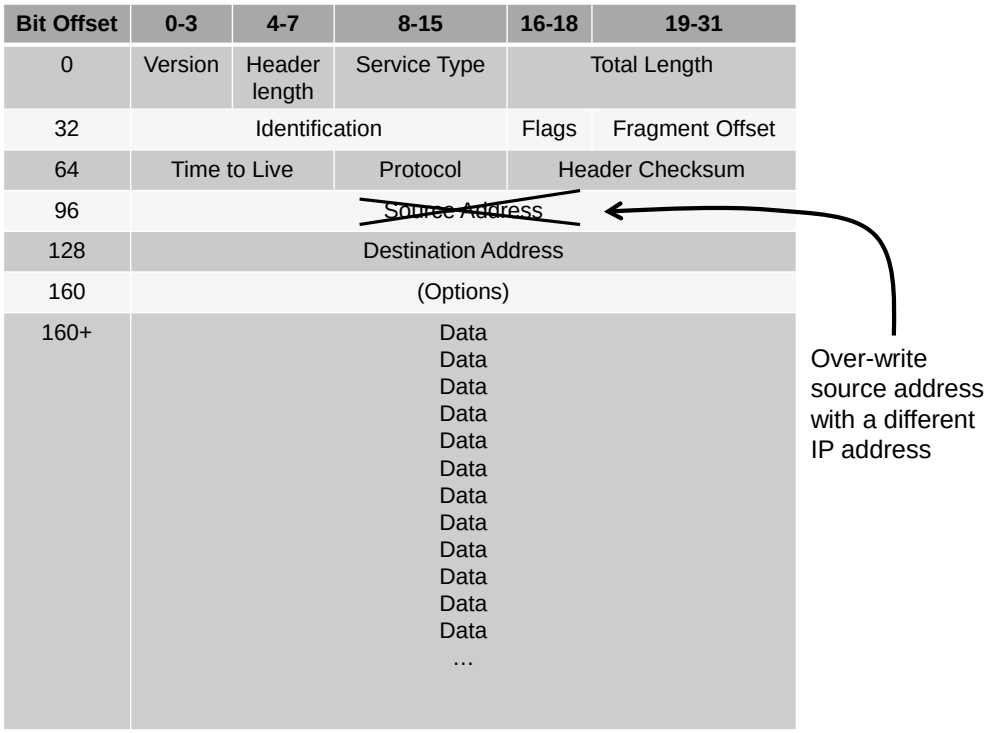

Figure 5.12: How IP spoofing works. The source address in the header of an IP packet is simply overwritten with a different IP address from the actual source. Note the header checksum field also needs to be updated.

How IP Spoofing is Used in Other Attacks

If an attacker sends an IP packet with a spoofed source address, then he will not receive any response from the destination server. In fact, with a spoofed source IP address on an outbound packet, the machine with the spoofed IP address will receive any response from the destination server, not the attacker.

Therefore, if an attacker is using IP spoofing on his outbound packets, he must either not care about any responses for these packets or he has some other way of receiving responses. For example, in denial-of-service attacks (which are discussed in more detail in Section 5.5), the attacker doesn't want to receive any responses back—he just wants to overwhelm some other Internet host with data requests. Alternatively, in IP spoofing attacks that are designed for circumventing firewall policy (Section 6.2) or TCP session hijacking (Section 5.4.4), the attacker has another, nonstandard way of getting response packets.

Dealing with IP Spoofing

Although it cannot be prevented, there are a number of ways of dealing with IP spoofing. For example, border routers, which are routers that span two or more subnetworks, can be configured to block packets from outside their administrative domain that have source addresses from inside that domain. Such packets are likely to be spoofed so as to appear they are coming from inside the subnetwork, when in fact they are coming from outside the domain. Similarly, such routers can also block outgoing traffic with source addresses from outside the domain. Such packets could indicate that someone inside the subnetwork is trying to launch an attack that uses IP spoofing, so this type of blocking could be an indication that machines inside the subnetwork have been taken over by a malware attack or are otherwise controlled by malicious parties.

In addition, IP spoofing can be combated by implementing IP traceback techniques, as discussed in Section 5.5.5. IP traceback involves methods for tracing the path of a packet back to its actual source address. Given this information, requests can then be made to the various autonomous systems along this path to block packets from this location. The ISP controlling the actual source address can also be asked to block suspicious machines entirely until it is determined that they are clean of any malware or malicious users.

5.3.4 Packet Sniffing

Because most data payloads of IP packets are not encrypted, the Internet Protocol allows for some types of eavesdropping, further compromising confidentiality. In particular, it is possible to listen in on the traffic in a network that is intended for the Internet. This process is known as ***packet sniffing***, and can be performed independently of whether the packets are traveling via wireless Internet or through a wired Internet, provided the attacker resides on the same network segment.

As discussed in Section 5.2.1, when frames are transmitted over an Ethernet network, they are received by every device on the same network segment. Each network interface in this segment will normally compare the frame's destination MAC address with its own MAC address, and discard the frame if it doesn't match. If a network interface is operating in ***promiscuous*** mode, however, it will retain all frames and read their contents. Setting a network interface to promiscuous mode allows an attacker to examine all data transmitted over a particular network segment, potentially recovering sensitive information such as passwords and other data. Combined with network analysis tools such as Wireshark, this data can be extracted from the raw packets. (See Figure 5.13.)

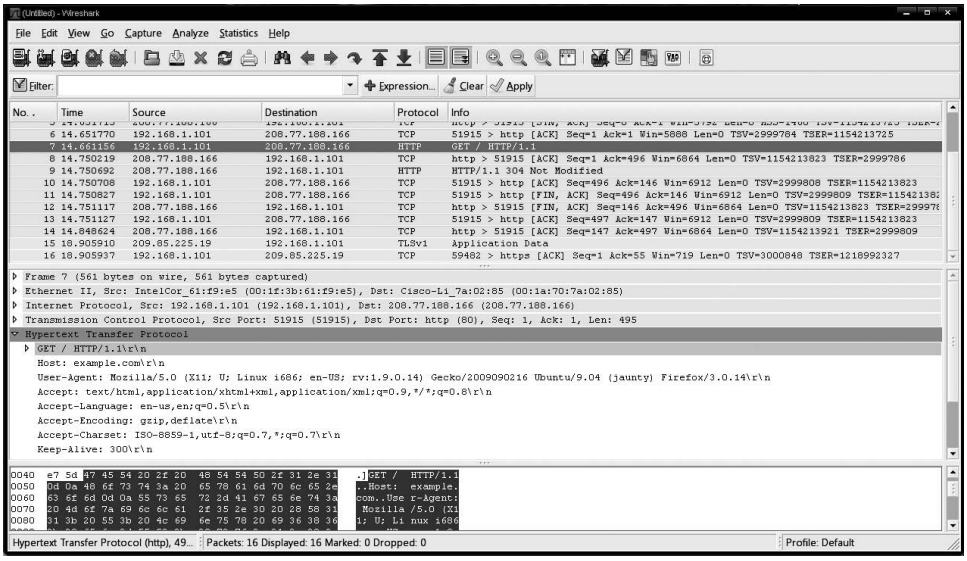

Figure 5.13: An example use of the Wireshark packet-sniffing tool. Here, the packet associated with an HTTP request to www.example.com has been captured and analyzed.

Defenses Against Packet Sniffing

Using a packet-sniffing tool such as Wireshark is not necessarily malicious. For instance, packet sniffing is commonly used to troubleshoot network-related problems or to determine if a computer is infected with adware or spyware (and is contacting outside IP addresses without the user's knowledge or consent). But packet sniffing can also be malicious, for instance if it is used to spy on unsuspecting members of a network.

There are several measures that can be put in place to prevent unwanted packet sniffing, besides the obvious precaution of preventing unauthorized access to a private network. For example, using Ethernet switches as opposed to hubs potentially reduces the number of machines on an attacker's network segment, which reduces the amount of traffic that may be sniffed. Note that there is no analog to the switch when communicating wirelessly, however. Since all wireless traffic is transmitted over the air, any device on the same wireless network may sniff traffic from any other device.

It may also be possible to detect when network devices are in promiscuous mode, although this has proven to be difficult in practice. One technique takes into account the fact that when a network interface is receiving all network traffic, the operating system behind that network interface is using much more processing power than if these frames were being dropped. Therefore, responses from that interface may be slightly delayed in comparison to those issued by interfaces not in promiscuous mode. Alternately, attempting to elicit responses to invalid packets from network devices may provide clues suggesting that a device is in promiscuous mode. For example, sending a packet to a suspected machine's IP address with a nonmatching MAC address would ordinarily be dropped by that network device, but if it is running in promiscuous mode, a response might be issued.

Despite these precautions and detection measures, packet sniffing remains a risk that should not be underestimated, especially on networks that may include malicious parties. To reduce the impact of packet sniffing, encryption mechanisms should be utilized in higher-level protocols to prevent attackers from recovering sensitive data. As an example, web traffic ordinarily contains an HTTP packet at the application layer, encapsulated in a TCP packet at the transport layer, and an IP packet at the network layer, and then an appropriate link layer frame such as Ethernet or 802.11 wireless. In a packet-sniffing scenario, an attacker can examine all HTTP content in an intercepted packet because no encryption is used at any layer. If the HTTPS protocol, which employs encryption at the application layer, is used instead, then even if an attacker sniffs traffic, the contents will be encrypted and so will be indecipherable to the attacker.

5.4 The Transport Layer

The transport layer builds on top of the network layer, which supports communication between machines, to provide for communication between processes. This extended addressing capability is achieved in the transport layer by viewing each machine (which has just one IP address) as having a collection of ports, each of which is capable of being the source or destination port for communication with a port on another machine. Indeed, the transport layer protocols for the Internet specify 16-bit source and destination port numbers in their headers. Each port is meant to be associated with a certain type of service offered by a host.

Two primary protocols operate at the transport layer for the Internet: the ***Transmission Control Protocol*** (***TCP***) and the ***User Datagram Protocol*** (***UDP***). TCP is the more sophisticated of these two and was defined together with IP as one of the original protocols for the Internet, which is why people sometimes refer to Internet protocols as "TCP/IP." TCP is used for some of the most fundamental operations of the Internet.

The main extra feature of TCP is that it is connection oriented and provides a reliable stream of bytes between two communicating parties with a guarantee that information arrives intact and in order. If a packet in such a stream is lost, TCP guarantees that it will be resent, so that there is no actual loss of data. Thus, TCP is the preferred protocol for transferring files, web pages, and email.

UDP, on the other hand, provides a best-effort communication channel between two ports. It is used primarily for applications where communication speed is more important than completeness, such as in a voice-over-IP conversation, where short drops are acceptable (as one might get from one lost packet), but not long pauses (as one might get from waiting for a lost packet to be resent).

5.4.1 Transmission Control Protocol (TCP)

TCP is a critical protocol for the Internet, since it takes the IP protocol, which routes packets between machines in a best effort fashion, and creates a protocol that can guarantee transmission of a stream of bits between two virtual ports. If a process needs to send a complete file to another computer, for instance, rather than do the hard work of chopping it into IP packets, sending them to the other machine, double-checking that all the packets made it intact, and resending any that were lost, the process can simply delegate the entire transfer to TCP. TCP takes care of all of these details.

TCP Features

A TCP session starts out by establishing a communication connection between the sender and receiver. Once a connection has been created, the parties can then communicate over the established channel. TCP ensures reliable transmission by using a sequence number that is initialized during the three-way handshake. Each subsequent transmission features an incremented sequence number, so that each party is aware when packets arrive out of order or not at all.

TCP also incorporates a cumulative acknowledgment scheme. Consider two TCP sessions, a sender and a receiver, communicating via their established TCP connection. After the sender sends the receiver a specified amount of data, the receiver will confirm that it has received the data by sending a response packet to the sender with the acknowledgment field set to the next sequence number it expects to receive. If any information has been lost, then the sender will retransmit it.

TCP also manages the amount of data that can be sent by one party while avoiding overwhelming the processing resources of the other or the bandwidth of the network itself, which is a concept known as ***flow control***. In particular, to efficiently manage flow control, TCP uses a technique known as a ***sliding window protocol***. Consider again two parties in a TCP conversation, a sender and receiver. With each packet, the receiver informs the sender of the size of the ***receive window***, which is the number of bytes of data it is willing to accept before the sender must pause and wait for a response, indicating the receiver is ready to accept more data. The sender also keeps track of the value of the last acknowledgment sent by the receiver. When sending data, the sender checks the sequence number of the packet to be sent, and only continues sending if this number is less than the last acknowledgment number plus the current size of the receive window (i.e., the sequence number falls within the current window of acceptable sequence numbers). Otherwise, it waits for an acknowledgment, at which point it adjusts its stored acknowledgment number, shifting the "sliding window" of sequence numbers. During the process of sending data, the sender sets a timer so that if no acknowledgment is received before the timer expires, the sender assumes data loss and retransmits.

In addition to managing data flow, TCP supports a checksum field to ensure correctness of data. TCP's checksum is not intended to be cryptographically secure, but rather is meant to detect inconsistencies in data due to network errors rather than malicious tampering. This checksum is typically supplemented by an additional checksum at the link layer, such as Ethernet, which uses the CRC-32 checksum.

Congestion Control

TCP tackles a final networking problem by implementing ***congestion control***: an attempt to prevent overwhelming a network with traffic, which would result in poor transmission rates and dropped packets. Congestion control is not implemented into TCP packets specifically, but rather is based on information gathered by keeping track of acknowledgments for previously sent data and the time required for certain operations. TCP adjusts data transmission rates using this information to prevent network congestion.

TCP Packet Format

The format of a TCP packet is depicted in Figure 5.14. Note that it includes source and destination ports, which define the communication connection for this packet and others like it. In TCP, connection sessions are maintained beyond the life of a single packet, so TCP connections have a ***state***, which defines the status of the connection. In the course of a TCP communication session, this state goes from states used to open a connection, to those used to exchange data and acknowledgments, to those used to close a connection.

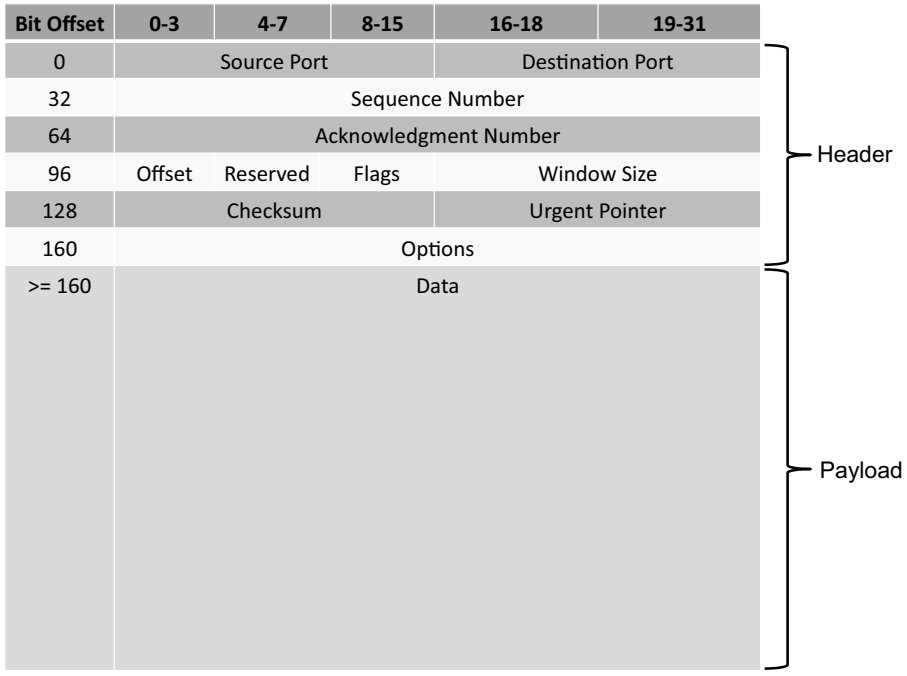

Figure 5.14: Format of a TCP packet.

TCP Connections

TCP uses a three-way handshake to establish a reliable connection stream between two parties, as depicted in Figure 5.15. First, a client sends a packet to the desired destination with the SYN flag (short for "synchronization") set. This packet includes a random initialization for a ***sequence number***, which is used to ensure reliable ordering of future data transmissions. In response, the server replies with a packet marked with both the SYN and ACK (short for "acknowledgment") flags, known as a SYN-ACK packet, indicating that the server wishes to accept the connection. This packet includes an ***acknowledgment number***, which is set to one more than the received sequence number, and a new random sequence number. Finally, the client responds with an ACK packet to indicate a successful connection has been established. The final ACK packet features an acknowledgment number set to one more than the most recently received sequence number, and the sequence number set to the recently received acknowledgment number. These choices are meant to defeat attacks against TCP based on predicting initial sequence numbers, which are discussed in Section 5.4.4.

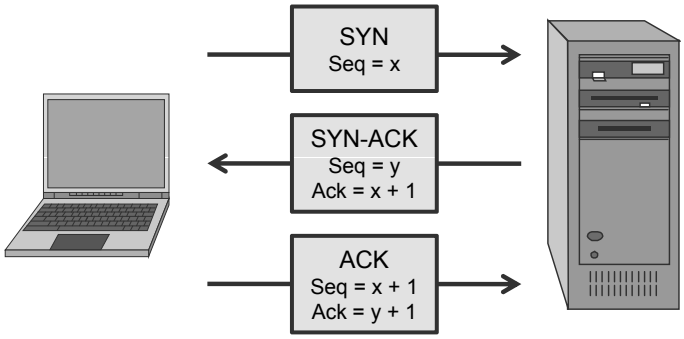

Figure 5.15: The three-way TCP handshake.

As mentioned above, TCP uses the notion of 16-bit ***port*** numbers, which differentiate multiple TCP connections. TCP packets include both a source port (the port from which the packet originated) and a destination port (the port where the packet will be received). Ports may range from 1 to 65,535 ($2^{16} - 1$), with lower port numbers being reserved for commonly used protocols and services. For example, port 80 is the default for the HTTP protocol, while ports 21 and 22 are reserved for FTP and SSH, respectively.

Most applications create network connections using ***sockets***, an abstraction that allows developers to treat network connections as if they were files. Developers simply read and write information as needed, while the operating system handles encapsulating this application-layer information in the lower levels of the TCP/IP stack.

5.4.2 User Datagram Protocol (UDP)

In contrast to TCP, the UDP protocol does not make a guarantee about the order or correctness of its packet delivery. It has no initial handshake to establish a connection, but rather allows parties to send messages, known as ***datagrams***, immediately. If a sender wants to communicate via UDP, it need only use a socket (defined with respect to a port on a receiver) and start sending datagrams, with no elaborate setup needed.

While UDP features a 16-bit checksum to verify the integrity of each individual packet, there is no sequence number scheme, so transmissions can arrive out of order or may not arrive at all. It is assumed that checking for missing packets in a sequence of datagrams is left to applications processing these packets. As a result, UDP can be much faster than TCP, which often requires retransmissions and delaying of packets.

UDP is often used in time-sensitive applications where data integrity is not as important as speed, such as DNS and Voice over IP (VoIP). In contrast, TCP is used for applications where data order and data integrity is important, such as HTTP, SSH, and FTP. The format of a UDP packet is depicted in Figure 5.16. Notice how much simpler it is than a TCP packet.

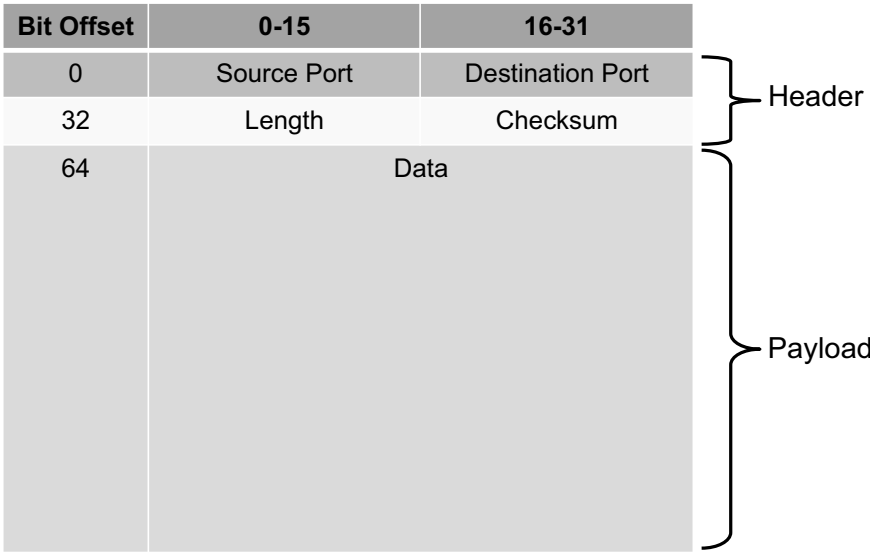

Figure 5.16: Format of a UDP packet.

5.4.3 Network Address Translation (NAT)

When people add computers, printers, and other network devices to their home networks, they typically do not buy new IP addresses and setup the new addresses directly on the Internet. Instead, they use ***network address translation*** (***NAT***), which allows all the machines on a local-area network to share a single public IP address. This public IP address represents the point of contact with the Internet for the entire LAN, while machines on the network have private IP addresses that are only accessible from within the network.

Since NAT allows an entire network to be assigned a single public IP address, widespread use of NAT has significantly delayed the inevitable exhaustion of the IPv4 address space. In fact, there is a lot of address capacity for NAT, because there are a number of private IP addresses that such networks are allowed to use which cannot be used on the (public) Internet. The private IP address are of the form 192.168.x.x, 172.16.x.x through 172.31.x.x, and 10.x.x.x. Thus, a NAT router represents the gateway between private IP addresses and the public Internet, and this router is responsible for managing both inbound and outbound Internet traffic. (See Figure 5.17.)

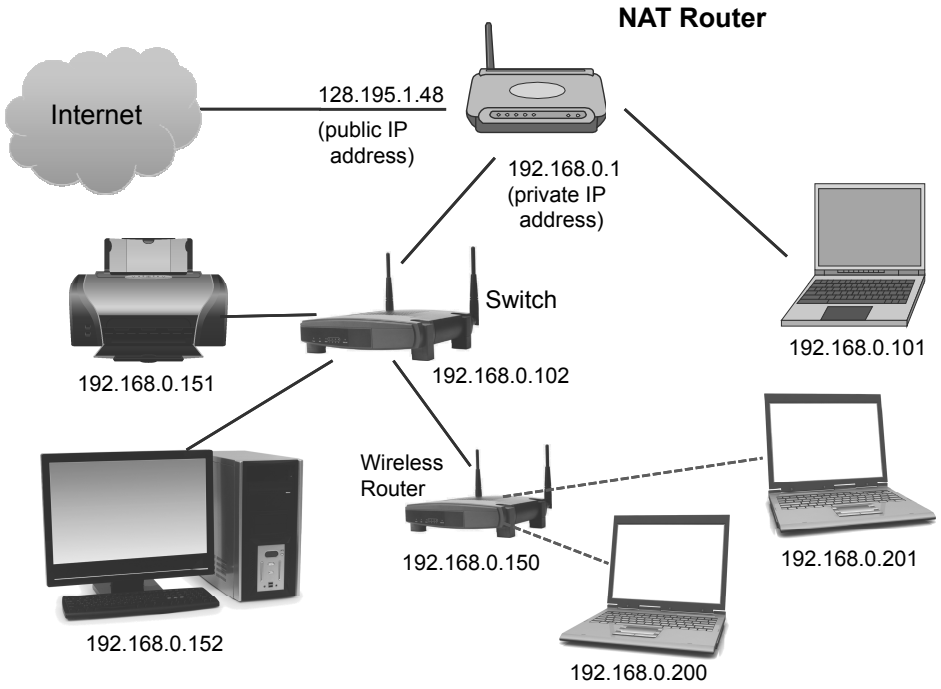

Figure 5.17: An example home network setup using a NAT router.

How NAT Works

To translate between private and public IP addresses, the NAT router maintains a lookup table that contains entries of the following form:

(private source IP, private source port, destination IP, public source port)

A NAT router dynamically rewrites the headers of all inbound and outbound TCP and UDP packets as follows. When a machine on the internal network attempts to send a packet to an external IP address, the NAT router creates a new entry in the lookup table associated with the source machine's private IP address and the internal source port of the transmitted packet. Next, it rewrites the source IP address to be that of the NAT device's public IP, opens a new public source port, and rewrites the IP header's source port field to contain the newly opened port. This public port and the destination IP address are recorded alongside the private source IP and private internal port in the NAT device's lookup table. The NAT device also adjusts any checksums contained in the packet, including those used by IP and TCP/UDP, to reflect the changes made. The packet is then forwarded to its destination.

On receiving a response, the NAT router checks its lookup table for any entries whose public source port corresponds to the destination port of the inbound packet and whose destination IP address (recorded because of the previous outbound packet) corresponds to the source IP of the inbound packet. Finally, the NAT router rewrites the IP headers of the inbound packet according to the lookup table, so that the packet is forwarded to the correct private IP address and private port.

This process effectively manages outbound traffic, but places several restrictions on the possibilities for inbound traffic. An external machine has no way of initiating a connection with a machine on the private network, since the internal machine does not have a publicly accessible IP address. This can actually be seen as a security feature, since no inbound traffic from the Internet can reach the internal network. Thus, in many ways, NAT devices can function as firewalls (Section 6.2), blocking risky contact from the external Internet.

Network Address Translation is not a perfect solution. In fact, it violates the ideal goal of ***end-to-end connectivity*** for machines on the Internet by not allowing direct communication between internal and external parties. In addition, NAT may cause problems when using several protocols, especially those using something other than TCP or UDP as a transport-layer protocol. Still, NAT has been crucial in delaying the exhaustion of the IPv4 address space and simplifying home networking.

5.4.4 TCP Session Hijacking

Let us now discuss a transport-layer security concern—TCP session hijacking—which is a way for an attacker to hijack or alter a TCP connection from another user. Such attacks come in several flavors, depending on the location and knowledge of the attacker.

TCP Sequence Prediction

The first type of session hijacking we discuss is a type of ***session spoofing***, since it creates a spoofed TCP session rather than stealing an existing one, but we still think of it as a type of session hijacking. Recall that TCP connections are initiated by a three-way handshake, in which the client sends a packet with the SYN flag sent, the server replies with a packet containing an initial sequence number and both the SYN and ACK flags set, and the client concludes by sending a packet with the received sequence number incremented by 1 and the ACK flag set. A ***TCP sequence prediction*** attack attempts to guess an initial sequence number sent by the server at the start of a TCP session, so as to create a spoofed TCP session.

Early TCP stacks implemented sequence numbers by using a simple counter that was incremented by 1 with each transmission. Without using any randomness, it was trivial to predict the next sequence number, which is the key to this attack. Modern TCP stack implementations use pseudo-random number generators to determine sequence numbers, which makes a TCP sequence prediction attack more difficult, but not impossible. A possible scenario might proceed as follows:

1. The attacker launches a denial-of-service attack against the client victim to prevent that client from interfering with the attack.

2. The attacker sends a SYN packet to the target server, spoofing the source IP address to be that of the client victim.

3. After waiting a short period of time for the server to send a reply to the client (which is not visible to the attacker and is not acted on by the client due to the DOS attack), the attacker concludes the TCP handshake by sending an ACK packet with the sequence number set to a prediction of the next expected number (based on information gathered by other means), again spoofing the source IP to be that of the client victim.

4. The attacker can now send requests to the server as if he is the client victim.

Blind Injection

Note that the above attack only allows one-way communication, since the attacker cannot receive any replies from the server due to the use of IP spoofing. Nevertheless, this method may allow an attacker to subvert a system that executes certain commands based on the source IP address of the requester. Indeed, Kevin Mitnick is said to have used this attack in 1995 for such a purpose. This type of attack is known as a ***blind injection***, because it is done without anticipating being able to see the server's response. Alternatively, it may be possible to inject a packet containing a command that creates a connection back to the attacker.

ACK Storms

A possible side-effect of a blind injection attack is that it can cause a client and server to become out-of-synchronization with respect to sequence numbers, since the server got a synchronized message the client never actually sent. TCP incorporates a method for clients and servers to become resynchronized when they get out of step, but it doesn't easily tolerate the kind of desynchronization that happens after a blind injection attack. So, after such an attack, the client and server might start sending ACK messages to each other, each trying to tell the other to start using "correct" sequence numbers. This back-and-forth communication is known as an ***ACK storm***, and it can continue until one of these messages is lost by accident or a firewall detects an ACK storm in progress and discards a bad ACK message.

Complete Session Hijacking

When an attacker is on the same network segment as the target server and/or client, an attacker can completely hijack an existing TCP session. This attack is possible because an attacker can use packet sniffing to see the sequence numbers of the packets used to establish the session. Given this information, an attacker can inject a packet with a highly probable sequence number (and a well-chosen attack command) to the server using a spoofed source IP address impersonating the client.

If used in combination with other network attacks, the possibility of an attacker who is in the same network segment as the target server and/or client victim allows for an even stronger type of session hijacking attack. In particular, an attacker on the same network segment as the client and/or server can use packet sniffing to see the sequence numbers of the packets used to establish a TCP session, as in a complete session-stealing attack. But he can also sometimes go a step further, by creating a man-in-the-middle

situation, e.g., using the ARP spoofing method discussed in Section 5.2.3. Once a man-in-the-middle scenario is in place, the attacker can then perform all subsequent actions as if he were the user he is masquerading as (by spoofed IP source addresses), and he can intercept all responses from both sides. (See Figure 5.18.)

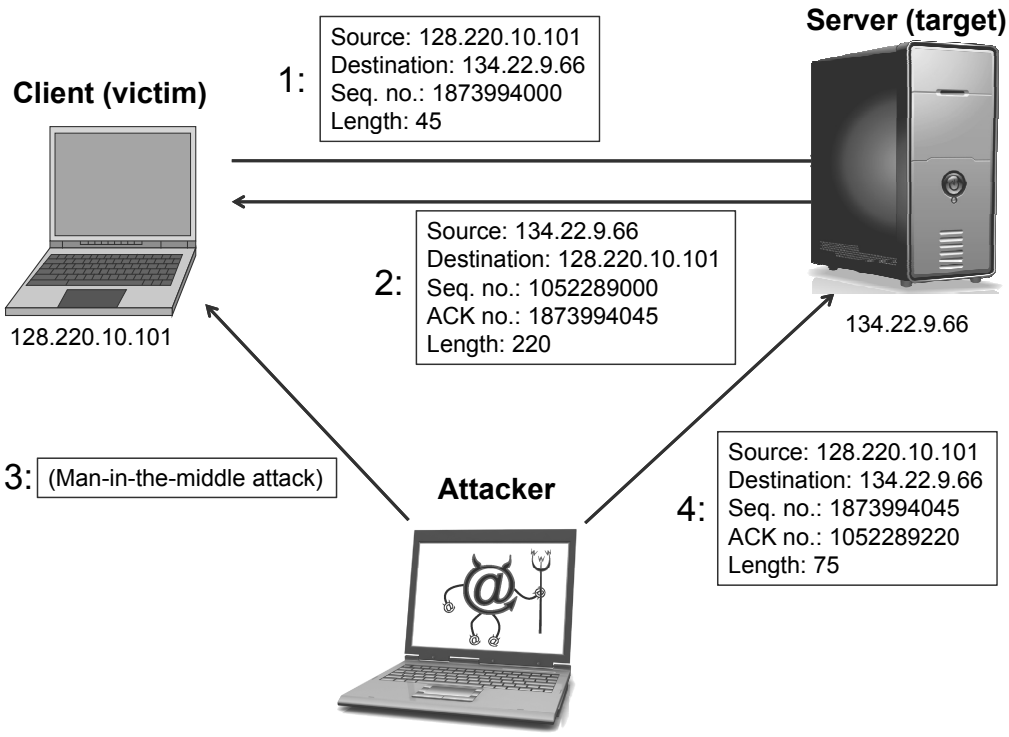

Figure 5.18: A TCP session hijacking attack.

Countermeasures

Countermeasures to TCP session hijacking attacks involve the use of encryption and authentication, either at the network layer, such as using IPsec (Section 6.3.2), or at the application layer, such as using application-layer protocols that encrypt entire sessions. In addition, web sites should avoid creating sessions that begin with secure authentication measures but subsequently switch over to unencrypted exchanges. Such sessions trade off efficiency for security, because they create a risk with respect to a TCP session hijacking attack.

5.5 Denial-of-Service Attacks

Because bandwidth in a network is finite, the number of connections a web server can maintain to clients is limited. Each connection to a server needs a minimum amount of network capacity to function. When a server has used up its bandwidth or the ability of its processors to respond to requests, then additional attempted connections are dropped and some potential clients will be unable to access the resources provided by the server. Any attack that is designed to cause a machine or piece of software to be unavailable and unable to perform its basic functionality is known as a ***denial-of-service*** (***DOS***) attack. This includes any situation that causes a server to not function properly, but most often refers to deliberate attempts to exceed the maximum available bandwidth of a server.

Because attackers in a DOS attack are not concerned with receiving any responses from a target, spoofing the source IP address is commonly used to obscure the identity of the attacker as well as make mitigation of the attack more difficult. Because some servers may stop DOS attacks by dropping all packets from certain blacklisted IP addresses, attackers can generate a unique source IP address for every packet sent, preventing the target from successfully identifying and blocking the attacker. This use of IP spoofing therefore makes it more difficult to target the source of a DOS attack. Before we can discuss countermeasures to DOS attacks, however, let us discuss some of the different kinds of network-based DOS attacks.

5.5.1 ICMP Attacks

Two simple DOS attacks, ping flood and smurf, exploit ICMP.

The Ping Flood Attack

As discussed in Section 5.3.2, the ping utility sends an ICMP echo request to a host, which in turn replies with an ICMP echo response. Normally, ping is used as a simple way to see if a host is working properly, but in a ***ping flood*** attack, a powerful machine can perform a DOS attack on a weaker machine. To carry out the attack, a powerful machine sends a massive amounts of echo requests to a single victim server. If the attacker can create many more ping requests than the victim can process, and the victim has enough network bandwidth to receive all these requests, then the victim server will be overwhelmed with the traffic and start to drop legitimate connections.

The Smurf Attack

A clever variation on this technique that takes advantage of misconfigured networks is known as a ***smurf attack***. Many networks feature a ***broadcast*** address by which a user can send a packet that is received by every IP address on the network. Smurf attacks exploit this property by sending ICMP packets with a source address set to the target and with a destination address set to the broadcast address of a network.

Once sent, each packet is received by every machine on the network, at which point every machine sends a reply ICMP packet to the indicated source address of the target. This results in an amplification effect that multiplies the number of packets sent by the number of machines on the network. In these attacks, the victim may be on the exploited network, or may be an entirely remote target, in which case the identity of the attacker is further obscured. An example of a smurf attack is depicted in Figure 5.19.

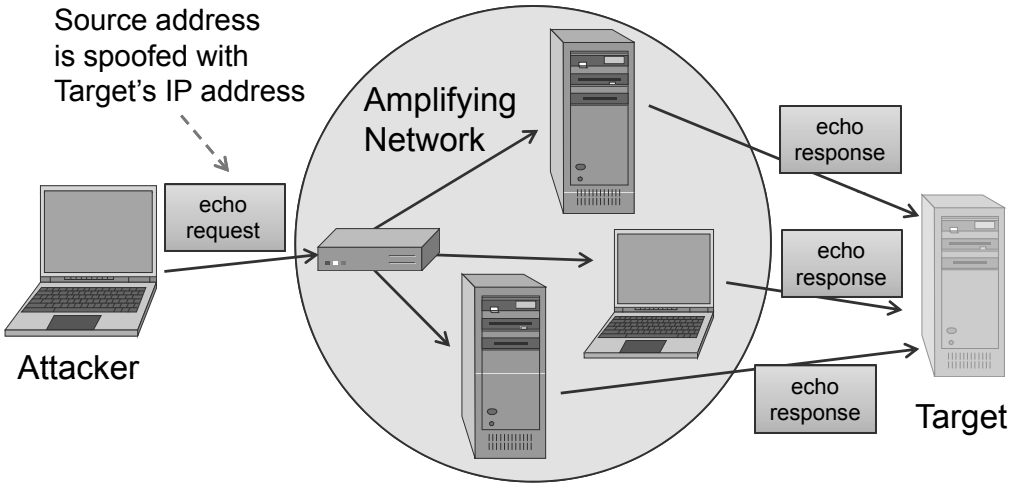

Figure 5.19: A smurf attack uses a misconfigured network to amplify traffic intended to overwhelm the bandwidth of a target.

To prevent smurf attacks, administrators should configure hosts and routers on their networks to ignore broadcast requests. In addition, routers should be configured to avoid forwarding packets directed to broadcast addresses, as this poses a security risk in that the network can be used as a ping flood amplifier. Finally, if a server is relatively weak, it would be wise for it to ignore ping requests altogether, to avoid ping floods.

5.5.2 SYN Flood Attacks

Another type of denial-of-service attack is known as a ***SYN flood*** attack. Recall (from Section 5.4.1) that to initiate a TCP session, a client first sends a SYN packet to a server, in response to which the server sends a SYN/ACK packet. This exchange is normally then followed by the client sending a concluding ACK packet to the server. If the client never sends the concluding ACK, however, the server waits for a certain time-out period and then discards the session.

How a SYN Flood Attack Works

In the SYN flood attack, an attacker sends a large number of SYN packets to the server, ignores the SYN/ACK replies, and never sends the expected ACK packets. In fact, an attacker initiating this attack in practice will probably use random spoofed source addresses in the SYN packets he sends, so that the SYN/ACK replies are sent to random IP addresses. If an attacker sends a large amount of SYN packets with no corresponding ACK packets, the server's memory will fill up with sequence numbers that it is remembering in order to match up TCP sessions with expected ACK packets. These ACK packets will never arrive, so this wasted memory ultimately blocks out other, legitimate TCP session requests.

Defenses Against SYN Flood Attacks

One commonly used technique to prevent SYN flooding features a mechanism known as ***SYN cookies***, which is credited to Daniel Bernstein. When SYN cookies are implemented, rather than dropping connections because its memory is filled, the server sends a specially crafted SYN/ACK packet without creating a corresponding memory entry. In this response packet, the server encodes information in the TCP sequence number as follows:

- The first 5 bits are a timestamp realized as a counter incremented every minute modulo 32.
- The next 3 bits are an encoded value representing the maximum segment size of transmission.
- The final 24 bits are a MAC (Section 1.3.4) of the server and client IP addresses, the server and client port numbers, and the previously used timestamp, computed using a secret key.

How SYN Cookies Work

According to the TCP specification, a legitimate client must reply with a sequence number equal to the previously sent sequence number plus 1. Therefore, when a client replies with an ACK packet, the server subtracts 1 to obtain the previously sent sequence number. It then compares the first 5 bits with the current timestamp to check if the connection has expired. Next, the server recomputes the 24-bit MAC using known IP and port information and compares with the value encoded in the sequence number. Finally, the server decodes the middle 3 bits to finish reconstructing the SYN queue entry, at which point the TCP connection can continue. If everything checks out with this ***SYN cookie check***, then the server initiates the TCP session.

SYN Cookies Limitations

At the time of this writing, Windows has not adopted SYN cookies, but they are implemented in several Linux distributions. Lack of widespread adoption may be due to some limitations introduced by the use of SYN cookies:

- The maximum segment size can only be eight possible values, since this is the most information that can be encoded in 3 bits.

- SYN cookies do not ordinarily allow the use of the TCP options field, since this information is usually stored alongside SYN queue entries.

Recent Linux SYN cookie implementations attempt to address this second limitation by encoding TCP option information in the timestamp field of TCP packets. Nevertheless, the inability to use several TCP options, many of which have become commonplace in the years since the initial development of SYN cookies, has made SYN cookies an unacceptable option in some situations.

Alternatives to SYN Cookies

As an alternative, techniques have been developed to more effectively manage half-opened connections, including implementing a special queue for half-open connections and not allocating resources for a TCP connection until an ACK packet has been received. These techniques are currently implemented in Windows.

5.5.3 Optimistic TCP ACK Attack

As mentioned in Section 5.4.1, the number of TCP packets allowed to be outstanding during a TCP communication session before requiring an ACK is known as a congestion window. As a server receives ACKs from a client, it dynamically adjusts the congestion window size, w, to reflect the estimated bandwidth available.

The window size grows when ACKs are received, and shrinks when segments arrive out of order or are not received at all, indicating missing data. In so doing, TCP helps to keep network congestion down while also trying to push data through the Internet as quickly as possible without overloading the capacity of the routers along the path that the packets are traveling. This congestion-control nature of TCP automatically adjusts as network conditions change, shrinking the congestion window when packets are lost and increasing it when they are successfully acknowledged.

How the Optimistic TCP ACK Attack Works

An ***optimistic TCP ACK attack*** is a denial-of-service attack that makes the congestion-control mechanism of TCP work against itself. In this attack, a rogue client tries to make a server increase its sending rate until it runs out of bandwidth and cannot effectively serve anyone else. If performed simultaneously against many servers, this attack can also create Internet-wide congestion by overwhelming the bandwidth resources of routers between the victims and attacker.

The attack is accomplished by the client sending ACKs to packets before they have been received to make the server increase its transmission speed. The aim of the client is to acknowledge "in-flight" packets, which have been sent by the server but have not yet been received by the client.

Defense Against the Optimistic TCP ACK Attack

While this attack has potentially serious impact, it has only rarely been performed in practice. Because the vulnerability is a consequence of the design of the TCP protocol itself, a true solution would require a redesign of TCP. Nevertheless, a real-life attack can be mitigated by implementing maximum traffic limits per client at the server level, and by promptly blocking traffic from clients whose traffic patterns indicate denial-of-service attempts. So it is not a major concern in practice.

5.5.4 Distributed Denial-of-Service

Today, most standard DOS attacks are impractical to execute from a single machine. Modern server technology allows web sites to handle an enormous amount of bandwidth—much greater than the bandwidth possible from a single machine. Nevertheless, denial-of-service conditions can still be created by using more than one attacking machine, in what is known as a ***distributed denial-of-service*** (***DDOS***) attack. In this attack, malicious users leverage the power of many machines (sometimes hundreds or even thousands) to direct traffic against a single web site in an attempt to create denial-of-service conditions. Major web sites, such as Yahoo!, Amazon, and Google, have been the targets of repeated DDOS attacks. Often, attackers carry out DDOS attacks by using botnets—large networks of machines that have been compromised and are controllable remotely. (See Figure 5.20.)

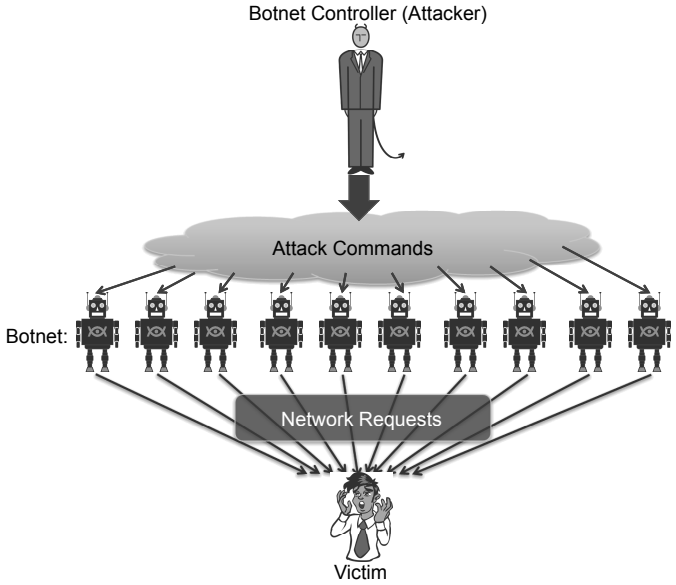

Figure 5.20: A botnet used to initiate a distributed denial-of-service attack.

In theory, there is no way to completely eliminate the possibility of a DDOS attack, since the bandwidth a server is able to provide its users will always be limited. Still, measures may be taken to mitigate the risks of DOS attacks. For example, many servers incorporate DOS protection mechanisms that analyze incoming traffic and drop packets from sources that are consuming too much bandwidth. Unfortunately, IP spoofing may make DDOS prevention more difficult, by obscuring the identity of the attacker bots and providing inconsistent information on where network traffic is coming from.

5.5.5 IP Traceback

In part prompted by the difficulties in determining the true origins of DDOS attacks featuring spoofed IP addresses, researchers have attempted to develop the concept of ***IP traceback***: determining the actual origin of a packet on the Internet, without relying on the IP source field contained in that potentially falsified packet.

Early IP traceback techniques relied on logging each packet forwarded by each router. While this approach may be effective, it places significant space requirements on routers, which may not have incentive to cooperate. A commonly proposed alternative relies on a technique known as ***packet marking***. In this approach, routers probabilistically or deterministically mark forwarded packets with information related to the path that packet has taken up to that point. Packet-marking schemes have an advantage in that once a victim has received enough packets to reconstruct a path to the attacker, no further cooperation is needed on the part of intermediate routers. A naive scheme would require each router to simply append its address to the end of a packet before forwarding it to the next router. While this approach has the advantage that a single packet carries all the information necessary to reconstruct a path to the attacker, it has a critical limitation in that it places unreasonably high overhead on routers, which must append data to every packet passing through. In addition, there is no mechanism to determine whether packets actually have the unused space necessary to record the complete path, besides inspecting the packet in-flight and possibly incurring further overhead by fragmenting the packet.

A more advanced approach to packet marking is known as ***node sampling***. Rather than encoding in each packet a list representing the entire path, a single field in the IP packet that has only enough room for one address is used. Each router overwrites this field of each packet with its own address with some probability p. Given enough packets marked in this way, a victim can use this field to determine each router traversed between the attacker and the victim. To reconstruct the path, note that in a large sample of marked packets, more packets will be marked with the addresses of routers that are closest to the victim. For example, the probability that a packet will be marked by the nearest router to the victim is p, the probability that a packet will be marked by the second-nearest router (and not overwritten by the nearest router) is $p * (1 - p)$, and so on. Therefore, by computing the expected number of marked packets for each network hop and correlating these figures with the proportions of packets retaining marks by each router, the path can be reconstructed.

Several other IP traceback techniques have been developed, including some that rely on the use of additional network protocols such as ICMP to relay path information. While many innovative schemes exist, few have been implemented in practice, in part due to the fact that these techniques require widespread cooperation from Internet routers. IP traceback is an example of a technique that attempts to solve the problem of authentication at the network layer. Protocol extensions such as IPsec (Section 6.3.2) and solutions such as Virtual Private Networking (Section 6.3.3) address the same problem by requiring cryptographic authentication for packets to verify their origin.

5.6 Exercises

For help with exercises, please visit **securitybook.net**.

Reinforcement

R-5.1 How many IP addresses are available under IPv6? Is it realistic to say that IPv6 will never run out of addresses?

R-5.2 What is the difference between a MAC address and an IP address?

R-5.3 Can two network interfaces have the same MAC address? Why or why not?

R-5.4 In the three-way handshake that initiates a TCP connection, if the SYN request has sequence number 156955003 and the SYN-ACK reply has sequence number 883790339, what are the sequence and acknowledgment numbers for the ACK response?

R-5.5 Can two network interfaces have the same IP address? Why or why not?

R-5.6 Show why installing static ARP tables on the machines of a local-area network does not prevent a malicious machine from intercepting traffic not intended for it.

R-5.7 Describe the difference between a switch, hub, and IP router, including their respective security implications.

R-5.8 What is an ACK storm and how does it start?

R-5.9 Jill lives in a large apartment complex and has a Wi-Fi access point that she keeps in her apartment. She likes her neighbors, so she doesn't put any password on her Wi-Fi and lets any of her neighbors use her Wi-Fi from their nearby apartments if they want to access the Internet. What kinds of security risks is Jill setting herself up for?

R-5.10 Explain how IP broadcast messages can be used to perform a smurf DOS attack.

R-5.11 Describe how sequence numbers are used in the TCP protocol. Why should the initial sequence numbers in the TCP handshake be randomly generated?

R-5.12 Why is it that packet sniffing can learn so much about the content of IP packets?

R-5.13 Explain why audio and video streams are typically transmitted over UDP instead of TCP.

R-5.14 TCP connections require a lot of overhead, as compared to UDP. Explain why web sites and file transfers are nevertheless typically transmitted over TCP instead of UDP.

R-5.15 How is it that a machine of a private network behind a NAT router can make a connection with a web server on the public Internet?

R-5.16 What is a distributed denial-of-service attack and how is it possible for a single person to orchestrate one?

Creativity

C-5.1 How many bytes are devoted to header and footer information (with respect to all layers of the IP protocol stack) of an Ethernet frame that contains a TCP packet inside it?

C-5.2 What is the absolute maximum number of IP addresses available under IPv4 if NAT is used to extend each one as much as possible?

C-5.3 Show how to extend the man-in-the-middle attack described in Section 5.2.3 to intercept all documents sent to a printer in a local-area network.

C-5.4 Suppose you suspect that your session with a server has been intercepted in a man-in-the-middle attack. You have a key, K, that you think you share with the server, but you might be only sharing it with an attacker. But the server also has a public key, K_P, which is widely known, and a private secret key, K_S, that goes with it. Describe how you can either confirm you share K with the server or discover that you share it only with a man-in-the-middle. Also, be sure your solution will not be discovered by a packet sniffer.

C-5.5 Explain how to use the three-way TCP handshake protocol to perform a distributed denial-of-service attack, such that the victim is any host computer and the "bots" that are bombarding the victim with packets are legitimate web servers.

C-5.6 Describe a data structure for keeping track of all open TCP connections for a machine. The data structure should support efficiently adding and deleting connections and searching by host, source port, and destination port.

C-5.7 Most modern TCP implementations use pseudo-random number generators (PRNG) to determine starting sequence numbers for TCP sessions. With such generators, it is difficult to compute the ith number generated, given only the $(i-1)$st number generated. Explain what network security risks are created if an attacker is able to break such a PRNG so that he can in fact easily compute the ith number generated, given only the $(i-1)$st number generated.

C-5.8 Either party in an established TCP session is allowed to instantly kill their session just by sending a packet that has the reset bit, RST, set to 1. After receiving such a packet, all other packets for this session are discarded and no further packets for this session are acknowledged. Explain how to use this fact in a way that allows a third party to kill an existing TCP connection between two others. This attack is called a ***TCP reset attack***. Include both the case where the third party can sniff packets from the existing TCP connection and the case where he cannot.

C-5.9 The TCP reset attack, described in the previous exercise, allows an ISP to easily shutdown any existing TCP session that connects a host in its network to another machine on the Internet. Describe some scenarios where it would be ethical and proper for an ISP to kill such a TCP session in this way and where it would not be ethical and proper to do so.

C-5.10 You are the system administrator for an provider that owns a large network (e.g., at least 64,000 IP addresses). Show how you can use SYN cookies to perform a DOS attack on a web server.

C-5.11 Show how to defend against the DOS attack of Exercise C-5.10.

C-5.12 Describe how to modify a NAT router to prevent packets with spoofed IP addresses from exiting a private network.

C-5.13 To defend against optimistic TCP ACK attacks, it has been suggested to modify the TCP implementation so that data segments are randomly dropped by the server. Show how this modification allows one to detect an optimistic ACK attacker.

C-5.14 You just got a call from the University system administrator, who says that either you or your roommate is issuing denial-of-service attacks against other students from your shared network segment. You know you are not doing this, but you are unsure about your roommate. How can you tell if this accusation is true or not? And if it is true, what should you do about it?

C-5.15 Johnny just set up a TCP connection with a web server in Chicago, Illinois, claiming he is coming in with a source IP address that clearly belongs to a network in Copenhagen, Denmark. In examining the session logs, you notice that he was able to complete the three-way handshake for this connection in 10 milliseconds. How can you use this information to prove Johnny is lying?

Projects

P-5.1 On an authorized virtual machine network, define three Linux virtual machines, Host A, Host B, and Attacker, which could in fact all really be on the same host computer. Let these machines be on the same LAN. On Attacker (using super-user privilege), write a simple sniffing tool to capture the packets going from Host A to Host B. Print out the header of the packets. The `pcap` library can be used to implement this tool.

P-5.2 On an authorized virtual machine network, define three virtual machines, Client, Server, Attacker, and Observer, which could in fact all really be on the same host computer. Using a packet-building tool, like Netwox, which can create TCP, UDP, or IP packets, have the Attacker perform an ARP spoofing attack on the Client, so that all traffic from the Server to the Client now goes to the Attacker. Have the Observer confirm the success of this attack using a packet sniffer.

P-5.3 On an authorized virtual machine network, define three virtual machines, Server, Attacker, and Observer, which could in fact all really be on the same host computer. Using a packet-building tool, like Netwox, which can create TCP, UDP, or IP packets, have the Attacker perform an SYN flood on the Server. Have the Observer confirm the success of this attack using a packet sniffer and failed attempts to establish TCP connections with the Server.

P-5.4 On an authorized virtual machine network, define three virtual machines, Client, Attacker, and Observer, which could in fact all really be on the same host computer. Using a packet-building tool, like Netwox, which can create TCP, UDP, or IP packets, have the Attacker sniff the packets from the Client and then perform an TCP reset attack (see Exercise C-5.8) on the Client. Have the Observer confirm the success of this attack using a packet sniffer while the Client is connected to a popular video-streaming web site on the Internet.

P-5.5 On an authorized virtual machine network, define four virtual machines, Client, Server, Attacker, and Observer, which could in fact all really be on the same host computer. Using a packet-building tool, like Netwox, which can create TCP, UDP, or IP packets, have the Attacker perform a TCP session hijacking attack on a TCP connection established between the Client and the Server. Test both the case when the Attacker can sniff packets from this communication and the case when he cannot (this latter case might seem difficult,

but with 32-bit sequence numbers it is not impossible). Have the Observer confirm the success or failure of this attack using a packet sniffer.

P-5.6 Design and implement a system to make a TCP/IP connection between two virtual machines on a virtual machine network.

P-5.7 Design and implement the software for a NAT router.

P-5.8 Working in a team of two or three people, find a Wi-Fi access point and access it with at least two laptop computers, one of which has packet-sniffing software installed. Take turns having one person access the Internet using various tools, such as browsers and email clients, and having another person watch their packets. Write a joint report describing the session, including the issues of privacy and security that it raises.

Chapter Notes

The books by Comer [18] and Tanenbaum [100] cover in detail computer networks and the protocols outlined in this chapter. Fundamentals of network security are presented in the books by Kaufman, Perlman, and Speciner [46] and by Stallings [96]. Authoritative references for Internet standards are the Request for Comments (RFC) documents by the Internet Engineering Task Force (IETF). Specifically, the network protocols mentioned in this chapter are described in the following RFCs:

- RFC 768: User Datagram Protocol (UDP)
- RFC 791: Internet Protocol (IP)
- RFC 792: Internet Control Message Protocol (ICMP)
- RFC 793: Transmission Control Protocol (TCP)
- RFC 826: Address Resolution Protocol (ARP)

Bellovin gives an overview of the vulnerabilities of the core Internet protocols [4]. The optimistic TCP acknowledgment attack is described in CERT vulnerability note VU#102014 and in the papers by Savage et al. [87] and by Sherwood et al. [93]. In particular, the defense mechanism described in Exercise C-5.13 is from [93].