2

A Security Review of Protocols: Lower Layers

In the next two chapters, we present an overview of the TCP/IP protocol suite. This chapter covers the lower layers and some basic infrastructure protocols, such as DNS; the next chapter discusses middleware and applications. Although we realize that this is familiar material to many people who read this book, we suggest that you *not* skip the chapter; our focus here is on security, so we discuss the protocols and areas of possible danger in that light.

A word of caution: A security-minded system administrator often has a completely different view of a network service than a user does. These two parties are often at opposite ends of the security/convenience balance. Our viewpoint is tilted toward one end of this balance.

2.1 Basic Protocols

TCP/IP is the usual shorthand for a collection of communications protocols that were originally developed under the auspices of the U.S. Defense Advanced Research Projects Agency (then *DARPA*, later *ARPA*, now DARPA again), and was deployed on the old ARPANET in 1983. The overview we can present here is necessarily sketchy. For a more thorough treatment, the reader is referred to any of a number of books, such as those by Comer [Comer, 2000; Comer and Stevens, 1998; Comer *et al.*, 2000], Kurose and Ross [2002], or Stevens [Stevens, 1995; Wright and Stevens, 1995; Stevens, 1996].

A schematic of the data flow is shown in Figure 2.1. Each row is a different *protocol layer*. The top layer contains the applications: mail transmission, login, video servers, and so on. These applications call the lower layers to fetch and deliver their data. In the middle of the spiderweb is the *Internet Protocol (IP)* [Postel, 1981b]. IP is a packet multiplexer. Messages from higher level protocols have an *IP header* prepended to them. They are then sent to the appropriate *device driver* for transmission. We will examine the IP layer first.
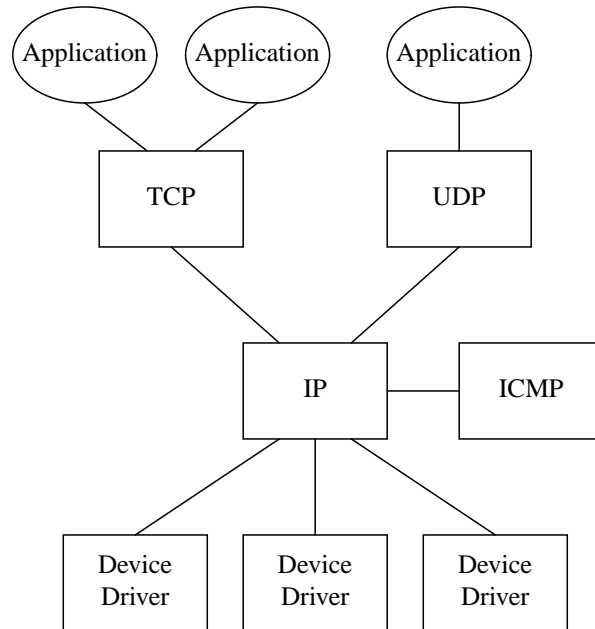
Figure 2.1: A schematic diagram of the different layers involving TCP/IP.

2.1.1 IP

IP *packets* are the bundles of data that form the foundation for the TCP/IP protocol suite. Every packet carries a source and destination address, some option bits, a header checksum, and a payload of data. A typical IP packet is a few hundred bytes long. These packets flow by the billions across the world over Ethernets, serial lines, SONET rings, packet radio connections, frame relay connections, *Asynchronous Transfer Mode (ATM)* links, and so on.

There is no notion of a *virtual circuit* or "phone call" at the IP level: every packet stands alone. IP is an unreliable *datagram* service. No guarantees are made that packets will be delivered, delivered only once, or delivered in any particular order. Nor is there any check for packet correctness. The checksum in the IP header covers only that header.

In fact, there is no guarantee that a packet was actually sent from the given source address. Any host can transmit a packet with any source address. Although many operating systems control this field and ensure that it leaves with a correct value, and although a few ISPs ensure that impossible packets do not leave a site [Ferguson and Senie, 2000], *you cannot rely on the validity of the source address, except under certain carefully controlled circumstances.* Therefore, authentication cannot rely on the source address field, although several protocols do just that. In general, attackers can send packets with faked return addresses: this is called *IP spoofing*. Authentication, and security in general, must use mechanisms in higher layers of the protocol.

A packet traveling a long distance will travel through many *hops*. Each hop terminates in a host or router, which forwards the packet to the next hop based on routing information. How a host or router determines the proper next hop is discussed in Section 2.2.1. (The approximate path to a given site can be discovered with the *traceroute* program. See Section 8.4.3 for details.)

Along the way, a router is allowed to drop packets without notice if there is too much traffic. Higher protocol layers (i.e., TCP) are supposed to deal with these problems and provide a reliable circuit to the application.

If a packet is too large for the next hop, it is *fragmented*. That is, it is divided into two or more packets, each of which has its own IP header, but only a portion of the payload. The fragments make their own separate ways to the ultimate destination. During the trip, fragments may be further fragmented. When the pieces arrive at the target machine, they are reassembled. As a rule, no reassembly is done at intermediate hops.

Some packet filters have been breached by being fed packets with pathological fragmentation [Ziemba *et al.*, 1995]. When important information is split between two packets, the filter can misprocess or simply pass the second packet. Worse yet, the rules for reassembly don't say what should happen if two overlapping fragments have different content. Perhaps a firewall will pass one harmless variant, only to find that the other dangerous variant is accepted by the destination host [Paxson, 1998]. (Most firewalls reassemble fragmented packets to examine their contents. This processing can also be a trouble spot.) Fragment sequences have also been chosen to tickle bugs in the IP reassembly routines on a host, causing crashes (see CERT Advisory CA-97.28).

IP Addresses

Addresses in IP version 4 (IPv4), the current version, are 32 bits long and are divided into two parts, a *network* portion and a *host* portion. The boundary is set administratively at each node, and in fact can vary within a site. (The older notion of fixed boundaries between the two address portions has been abandoned, and has been replaced by *Classless Inter-Domain Routing (CIDR)*. A CIDR network address is written as follows:

207.99.106.128/25

In this example, the first 25 bits are the network field (often called the *prefix*); the host field is the remaining seven bits.)

Host address portions of either all 0s or all 1s are reserved for broadcast addresses. A packet sent with a foreign network's broadcast address is known as a *directed broadcast*; these can be very dangerous, as they're a way to disrupt many different hosts with minimal effort. Directed broadcasts have been used by attackers; see Section 5.8 for details. Most routers will let you disable forwarding such packets; we strongly recommend this option.

People rarely use actual IP addresses: they prefer domain names. The name is usually translated by a special distributed database called the *Domain Name System*, discussed in Section 2.2.2.

2.1.2   ARP

IP packets are often sent over Ethernets. Ethernet devices do not understand the 32-bit IPv4 addresses: They transmit Ethernet packets with 48-bit Ethernet addresses. Therefore, an IP driver must translate an IP destination address into an Ethernet destination address. Although there are some static or algorithmic mappings between these two types of addresses, a table lookup is usually required. The *Address Resolution Protocol (ARP)* [Plummer, 1982] is used to determine these mappings. (ARP is used on some other link types as well; the prerequisite is some sort of link-level broadcast mechanism.)

ARP works by sending out an Ethernet broadcast packet containing the desired IP address. That destination host, or another system acting on its behalf, replies with a packet containing the IP and Ethernet address pair. This is cached by the sender to reduce unnecessary ARP traffic.

There is considerable risk here if untrusted nodes have write access to the local net. Such a machine could emit phony ARP queries or replies and divert all traffic to itself; it could then either impersonate some machines or simply modify the data streams *en passant*. This is called ARP *spoofing* and a number of *Hacker Off-the-Shelf (HOTS)* packages implement this attack.

The ARP mechanism is usually automatic. On special security networks, the ARP mappings may be statically hardwired, and the automatic protocol suppressed to prevent interference. If we absolutely never want two hosts to talk to each other, we can ensure that they don't have ARP translations (or have wrong ARP translations) for each other for an extra level of assurance. It can be hard to ensure that they never acquire the mappings, however.

2.1.3   TCP

The IP layer is free to drop, duplicate, or deliver packets out of order. It is up to the *Transmission Control Protocol (TCP)* [Postel, 1981c] layer to use this unreliable medium to provide reliable *virtual circuit*s to users' processes. The packets are shuffled around, retransmitted, and reassembled to match the original data stream on the other end.

The ordering is maintained by *sequence numbers* in every packet. Each byte sent, as well as the open and close requests, are numbered individually. A separate set of sequence numbers is used for each end of each connection to a host.

All packets, except for the very first TCP packet sent during a conversation, contain an *acknowledgment* number; it provides the sequence number of the next expected byte.

Every TCP message is marked as coming from a particular host and *port number*, and going to a destination host and port. The 4-tuple

$$\langle localhost, localport, remotehost, remoteport \rangle$$

uniquely identifies a particular circuit. It is not only permissible, it is quite common to have many different circuits on a machine with the same local port number; everything will behave properly as long as either the remote address or the port number differ.

Servers, processes that wish to provide some Internet service, *listen* on particular ports. By convention, server ports are low-numbered. This convention is not always honored, which can

cause security problems, as you'll see later. The port numbers for all of the standard services are assumed to be known to the caller. A listening portis in some sense half-open; only the local host and port number are known. (Strictly speaking, not even the local host address need be known. Computers can have more than one IP address, and connection requests can usually be addressed to any of the legal addresses for that machine.) When a connection request packet arrives, the other fields are filled in. If appropriate, the local operating system will clone the listening connection so that further requests for the same port may be honored as well.

Clients use the offered services. They connect from a local port to the appropriate server port. The local port is almost always selected at random by the operating system, though clients are allowed to select their own.

Most versions of TCP and UDP for UNIX systems enforce the rule that only the superuser (*root*) can create a port numbered less than 1024. These are *privileged ports*. The intent is that remote systems can trust the authenticity of information written to such ports. The restriction is a convention only, and is *not* required by the protocol specification. In any event, it is meaningless on non-UNIX operating systems. The implications are clear: One can trust the sanctity of the port number only if one is certain that the originating system has such a rule, is capable of enforcing it, and is administered properly. It is not safe to rely on this convention.

TCP Open

TCP open, a three-step process, is shown in Figure 2.2. After the server receives the initial SYN packet, the connection is in a *half–opened* state. The server replies with its own sequence number, and awaits an acknowledgment, the third and final packet of a TCP open.

Attackers have gamed this half-open state. SYN attacks (see Section 5.8.2) flood the server with the first packet only, hoping to swamp the host with half-open connections that will never be completed. In addition, the first part of this three-step process can be used to detect active TCP services without alerting the application programs, which usually aren't informed of incoming connections until the three-packet handshake is complete (see Section 6.3 for more details).

The sequence numbers mentioned earlier have another function. Because the initial sequence number for new connections changes constantly, it is possible for TCP to detect stale packets from previous incarnations of the same circuit (i.e., from previous uses of the same 4-tuple). There is also a modest security benefit: A connection cannot be fully established until both sides have acknowledged the other's initial sequence number.

But there is a threat lurking here. If an attacker can predict the target's choice of starting points—and Morris showed that this was indeed possible under certain circumstances [Morris, 1985; Bellovin, 1989]—then it is possible for the attacker to trick the target into believing that it is talking to a trusted machine. In that case, protocols that depend on the IP source address for authentication (e.g., the "*r*" commands discussed later) can be exploited to penetrate the target system. This is known as a *sequence number attack*.

Two further points are worth noting. First, Morris's attack depended in part on being able to create a legitimate connection to the target machine. If those are blocked, perhaps by a firewall, the attack would not succeed. Conversely, a gateway machine that extends too much trust to inside machines may be vulnerable, depending on the exact configuration involved. Second, the concept
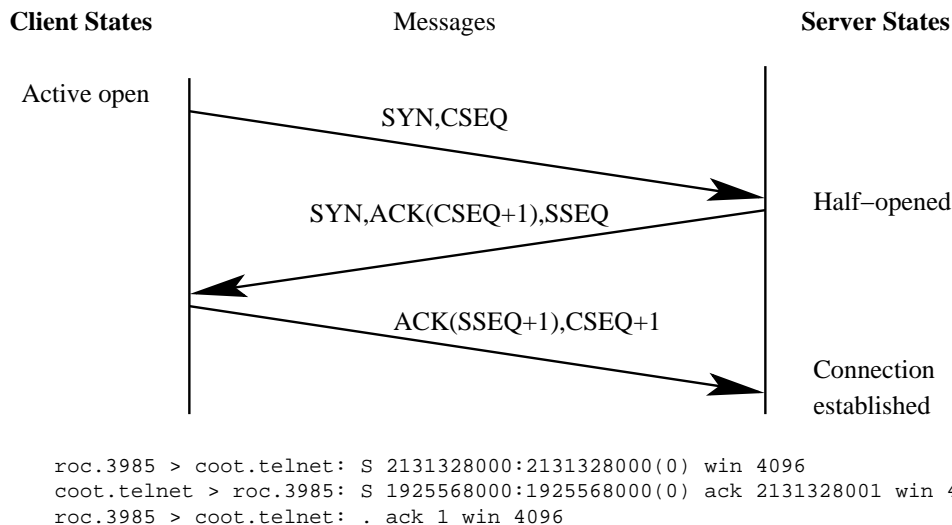
Client States	Messages	**Server States**

Active open

SYN,CSEQ

Half–opened

SYN,ACK(CSEQ+1),SSEQ

ACK(SSEQ+1),CSEQ+1

Connection
established

```
roc.3985 > coot.telnet: S 2131328000:2131328000(0) win 4096
coot.telnet > roc.3985: S 1925568000:1925568000(0) ack 2131328001 win 4096
roc.3985 > coot.telnet: . ack 1 win 4096
```

Figure 2.2: TCP Open The client sends the server a packet with the SYN bit set, and an initial client sequence number CSEQ. The server's reply packet has both the SYN and ACK packets set, and contains both the client's (plus 1) and server's sequence number (SSEQ) for this session. The client increments its sequence number, and replies with the ACK bit set. At this point, either side may send data to the other.

of a sequence number attack can be generalized. Many protocols other than TCP are vulnerable [Bellovin, 1989]. In fact, TCP's three-way handshake at connection establishment time provides more protection than do some other protocols. The hacker community started using this attack in late 1995 [Shimomura, 1996], and it is quite common now (see CERT Advisory CA-95.01 and CERT Advisory CA-96.21).

Many OS vendors have implemented various forms of randomization of the initial sequence number. The scheme described in [Bellovin, 1996] works; many other schemes are susceptible to statistical attacks (see CERT Advisory CA-2001-09). Michal Zalewski [2002] provided the clever visualizations of sequence number predictability shown in Figure 2.3. Simple patterns imply that the sequence number is easily predictable; diffuse clouds are what should be seen. It isn't that hard to get sequence number generation right, but as of this writing, most operating systems don't. With everything from cell phones to doorbells running an IP stack these days, perhaps it is time to update RFC 1123 [Braden, 1989a], including sample code, to get stuff like this right.

TCP Sessions

Once the TCP session is open, it's full-duplex: data flows in both directions. It's a pure stream, with no record boundaries. The implementation is free to divide user data among as many or as few packets as it chooses, without regard to the way in which the data was originally written by the user process. This behavior has caused trouble for some firewalls that assumed a certain packet structure.
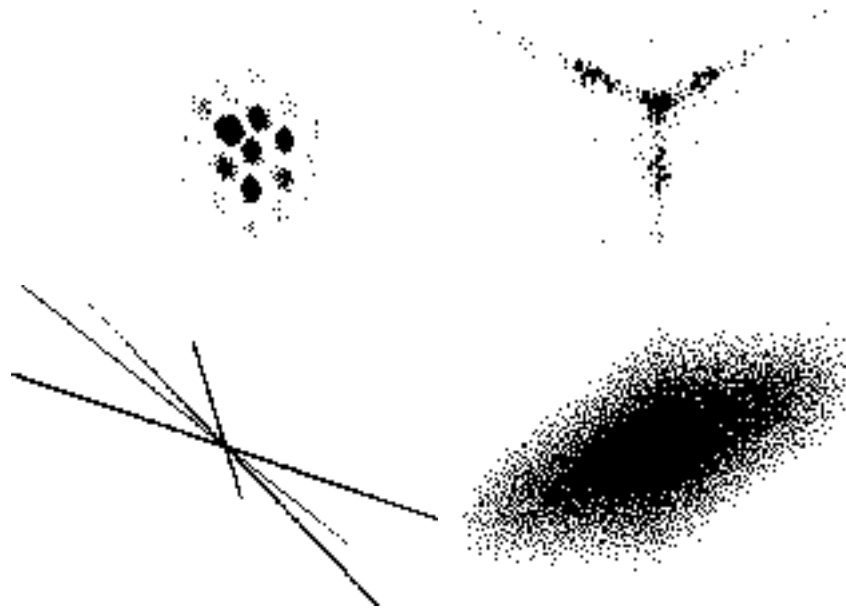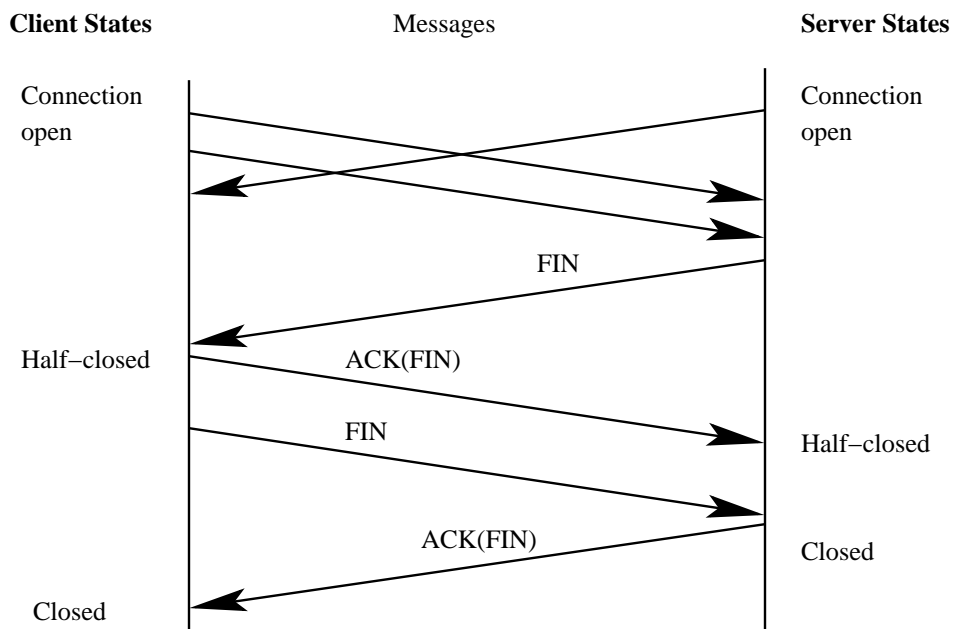
Figure 2.3: These are phase diagrams of the sequence number generators for four operating systems. The lower right shows a correct implementation of RFC 1948 sequence number generation (by FreeBSD 4.6.) The artistic patterns of the other three systems denote predictability that can be exploited by an attacker. The upper right shows IRIX 6.5.15m, the upper left Windows NT 4.0 SP3, and the lower left shows a few of the the many TCP/IP stacks for OpenVMS.

The TCP close sequence (see Figure 2.4) is asymmetric; each side must close its end of the connection independently.

2.1.4   SCTP

A new transport protocol, *Stream Control Transmission Protocol (SCTP)*, has recently been defined [Stewart *et al.*, 2000; Coene, 2002; Ong and Yoakum, 2002]. Like TCP, it provides reliable, sequenced delivery, but it has a number of other features.

The most notable new feature is the capability to multiplex several independent streams on a SCTP connection. Thus, a future FTP built on top of SCTP instead of TCP wouldn't need a PORT command to open a separate stream for the data channel. Other improvements include a four-way handshake at connection establishment time, to frustrate denial-of-service attacks, record-marking within each stream, optional unordered message delivery, and multi-homing of each connection. It's a promising protocol, though it isn't clear if it will catch on. Because it's new, not many firewalls support it yet. That is, not many firewalls provide the capability to filter SCTP traffic on a per-port basis, nor do they have any proxies for applications running on top of

Client States	Messages	**Server States**

Connection
open

Connection
open

FIN

Half−closed

ACK(FIN)

FIN

Half−closed

ACK(FIN)

Closed

Closed

Closed

```
coot.telnet > roc.3985: P 87:94(7) ack 45 win 4096
roc.3985 > coot.telnet: . ack 94 win 4096
roc.3985 > coot.telnet: P 45:46(1) ack 94 win 4096
coot.telnet > roc.3985: P 94:98(4) ack 46 win 4096
coot.telnet > roc.3985: F 98:98(0) ack 46 win 4096
roc.3985 > coot.telnet: . ack 99 win 4096
roc.3985 > coot.telnet: F 46:46(0) ack 99 win 4096
coot.telnet > roc.3985: . ack 47 win 4095
```

Figure 2.4: TCP I/O The TCP connection is full duplex. Each end sends a FIN packet when it is done transmitting, and the other end acknowledges. (All other packets here contain an ACK showing what has been received; those ACKs are omitted, except for the ACKs of the FINs.) A reset (RST) packet is sent when a protocol violation is detected and the connection needs to be torn down.

SCTP. Moreover, some of the new features, such as the capability to add new IP addresses to the connection dynamically, may pose some security issues. Keep a watchful eye on the evolution of SCTP; it was originally built for telephony signaling, and may become an important part of multimedia applications.

2.1.5 UDP

The *User Datagram Protocol (UDP)* [Postel, 1980] extends to application programs the same level of service used by IP. Delivery is on a best-effort basis; there is no error correction, retransmission, or lost, duplicated, or re-ordered packet detection. Even error detection is optional with UDP. Fragmented UDP packets are reassembled, however.

To compensate for these disadvantages, there is much less overhead. In particular, there is no connection setup. This makes UDP well suited to query/response applications, where the number of messages exchanged is small compared to the connection setup and teardown costs incurred by TCP.

When UDP is used for large transmissions, it tends to behave badly on a network. The protocol itself lacks flow control features, so it can swamp hosts and routers and cause extensive packet loss.

UDP uses the same port number and server conventions as does TCP, but in a separate address space. Similarly, servers usually (but not always) inhabit low-numbered ports. There is no notion of a circuit. All packets destined for a given port number are sent to the same process, regardless of the source address or port number.

It is much easier to spoof UDP packets than TCP packets, as there are no handshakes or sequence numbers. Extreme caution is therefore indicated when using the source address from any such packet. Applications that care *must* make their own arrangements for authentication.

2.1.6 ICMP

The *Internet Control Message Protocol (ICMP)* [Postel, 1981a] is the low-level mechanism used to influence the behavior of TCP and UDP connections. It can be used to inform hosts of a better route to a destination, to report trouble with a route, or to terminate a connection because of network problems. It is also a vital part of the two most important low-level monitoring tools for network administrators: *ping* and *traceroute* [Stevens, 1995].

Many ICMP messages received on a given host are specific to a particular connection or are triggered by a packet sent by that machine. The hacker community is fond of abusing ICMP to tear down connections. (Ask your Web search engine for `nuke.c`.)

Worse things can be done with `Redirect` messages. As explained in the following section, anyone who can tamper with your knowledge of the proper route to a destination can probably penetrate your machine. The `Redirect` messages should be obeyed only by hosts, not routers, and only when a message comes from a router on a directly attached network. However, not all routers (or, in some cases, their administrators) are that careful; it is sometimes possible to abuse ICMP to create new paths to a destination. If that happens, you are in serious trouble indeed.

Unfortunately, it is extremely inadvisable to block all ICMP messages at the firewall. Path MTU—the mechanism by which hosts learn how large a packet can be sent without fragmentation—requires that certain Destination Unreachable messages be allowed through [Mogul and Deering, 1990]. Specifically, it relies on ICMP Destination Unreachable, Code 4 messages: The packet is too large, but the "Don't Fragment" bit was set in the IP header. If you block these messages and some of your machines send large packets, you can end up with hard-to-diagnose dead spots. The risks notwithstanding, we strongly recommend permitting inbound Path MTU messages. (Note that things like IPsec tunnels and PPP over Ethernet, which is commonly used by DSL providers, can reduce the effective MTU of a link.)

IPv6 has its own version of ICMP [Conta and Deering, 1998]. ICMPv6 is similar in spirit, but is noticeably simpler; unused messages and options have been deleted, and things like Path MTU now have their own message type, which simplifies filtering.

2.2 Managing Addresses and Names

2.2.1 Routers and Routing Protocols

> "R$\overline{oo}'$•ting" is what fans do at a football game, what pigs do for truffles under oak trees in the Vaucluse, and what nursery workers intent on propagation do to cuttings from plants. "Rou$'$•ting" is how one creates a beveled edge on a tabletop or sends a corps of infantrymen into full-scale, disorganized retreat. Either pronunciation is correct for *routing*, which refers to the process of discovering, selecting, and employing paths from one place to another (or to many others) in a network.[1]

> *Open Systems Networking: TCP/IP and OSI*
> —DAVID M. PISCITELLO AND A. LYMAN CHAPIN

Routing protocols are mechanisms for the dynamic discovery of the proper paths through the Internet. They are fundamental to the operation of TCP/IP. Routing information establishes two paths: from the calling machine to the destination and back. The second path may or may not be the reverse of the first. When they aren't, it is called an *asymmetric route*. These are quite common on the Internet, and can cause trouble if you have more than one firewall (see Section 9.4.2). From a security perspective, it is the return path that is often more important. When a target machine is attacked, what path do the reverse-flowing packets take to the attacking host? If the enemy can somehow subvert the routing mechanisms, then the target can be fooled into believing that the enemy's machine is really a trusted machine. If that happens, authentication mechanisms that rely on source address verification will fail.

---

1. If you're talking to someone from Down Under, please pronounce it "Rou$'$•ting."

7 There are a number of ways to attack the standard routing facilities. The easiest is to employ the IP *loose source route* option. With it, the person initiating a TCP connection can specify an explicit path to the destination, overriding the usual route selection process. According to RFC 1122 [Braden, 1989b], the destination machine must use the inverse of that path as the return route, whether or not it makes any sense, which in turn means that an attacker can impersonate any machine that the target trusts.

The easiest way to defend against source routing problems is to reject packets containing the option. Many routers provide this facility. Source routing is rarely used for legitimate reasons, although those do exist. For example, it can be used for debugging certain network problems; indeed, many ISPs use this function on their backbones. You will do yourself little harm by disabling it at your firewall—the uses mentioned above rarely need to cross administrative boundaries. Alternatively, some versions of *rlogind* and *rshd* will reject connections with source routing present. This option is inferior because there may be other protocols with the same weakness, but without the same protection. Besides, one abuse of source routing—learning the sequence numbers of legitimate connections in order to launch a sequence-number guessing attack—works even if the packets are dropped by the application; the first response from TCP did the damage.

8 Another path attackers can take is to play games with the routing protocols themselves. For example, it is relatively easy to inject bogus *Routing Information Protocol (RIP)* [Malkin, 1994] packets into a network. Hosts and other routers will generally believe them. If the attacking machine is closer to the target than is the real source machine, it is easy to divert traffic. Many implementations of RIP will even accept host-specific routes, which are much harder to detect.

Some routing protocols, such as RIP version 2 [Malkin, 1994] and *Open Shortest Path First (OSPF)* [Moy, 1998], provide for an authentication field. These are of limited utility for three reasons. First, some sites use simple passwords for authentication, even though OSPF has stronger variants. Anyone who has the ability to play games with routing protocols is also capable of collecting passwords wandering by on the local Ethernet cable. Second, if a legitimate speaker in the routing dialog has been subverted, then its messages—correctly and legitimately signed by the proper source—cannot be trusted. Finally, in most routing protocols, each machine speaks only to its neighbors, and they will repeat what they are told, often uncritically. Deception thus spreads.

Not all routing protocols suffer from these defects. Those that involve dialogs between pairs of hosts are harder to subvert, although sequence number attacks, similar to those described earlier, may still succeed. A stronger defense is topological. Routers can and should be configured so that they know what routes can legally appear on a given wire. In general, this can be difficult to achieve, but firewall routers are ideally positioned to implement the scheme relatively simply. This can be hard if the routing tables are too large. Still, the general case of routing protocol security is a research question.

Some ISPs use OSI's IS-IS routing protocol internally, instead of OSPF. This has the advantage that customers can't inject false routing messages: IS-IS is not carried over IP, so there is no connectivity to customers. Note that this technique does not help protect against internal Bad Guys.

BGP

Border Gateway Protocol (BGP) distributes routing information over TCP connections between routers. It is normally run within or between ISPs, between an ISP and a multi-homed customer, and occasionally within a corporate intranet. The details of BGP are quite arcane, and well beyond the scope of this book—see [Stewart, 1999] for a good discussion. We can cover important security points here, however.

BGP is used to populate the routing tables for the core routers of the Internet. The various *Autonomous Systems (AS)* trade network location information via announcements. These announcements arrive in a steady stream, one every couple of seconds on average. It can take 20 minutes or more for an announcement to propagate through the entire core of the Internet. The path information distributed does not tell the whole story: There may be special arrangements for certain destinations or packet types, and other factors, such as route aggregation and forwarding delays, can muddle things.

Clearly, these announcements are vital, and incorrect announcements, intentional or otherwise, can disrupt some or even most of the Internet. Corrupt announcements can be used to perform a variety of attacks, and we probably haven't seen the worst of them yet. We have heard reports of evildoers playing BGP games, diverting packet flows via GRE tunnels (see Section 10.4.1) through convenient routers to eavesdrop on, hijack, or suppress Internet sessions. Others announce a route to their own network, attack a target, and then remove their route before forensic investigators can probe the source network.

ISPs have been dealing with routing problems since the beginning of time. Some BGP checks are easy: an ISP can filter announcements from its own customers. But the ISP cannot filter announcements from its peers—almost anything is legal. The infrastructure to fix this doesn't exist at the moment.

Theoretically, it is possible to hijack a BGP TCP session. MD5 BGP authentication can protect against this (see [Heffernan, 1998]) and is available, but it is not widely used. It should be.

Some proposals have been made to solve the problem [Kent *et al.*, 2000b, 2000a; Goodell *et al.*, 2003; Smith and Garcia-Luna-Aceves, 1996]. One proposal, S-BGP, provides for chains of digital signatures on the entire path received by a BGP speaker, all the way back to the origin. Several things, however, are standing in the way of deployment:

- Performance assumptions seem to be unreasonable for a busy router. A lot of public key cryptography is involved, which makes the protocol very compute-intensive. Some precomputation may help, but hardware assists may be necessary.

- A *Public Key Infrastructure (PKI)* based on authorized IP address assignments is needed, but doesn't exist.

- Some people have political concerns about the existence of a central routing registry. Some companies don't want to explicitly reveal peering arrangements and customer lists, which can be a target for salesmen from competing organizations.

For now, the best solution for end-users (and, for that matter, for ISPs) is to do regular *traceroute*s to destinations of interest, including the name servers for major zones. Although

Table 2.1: Some Important DNS Record Types

Type	Function
A	IPv4 address of a particular host
AAAA	IPv6 address of a host
NS	Name server. Delegates a subtree to another server.
SOA	Start of authority. Denotes start of subtree; contains cache and configuration parameters, and gives the address of the person responsible for the zone.
MX	Mail exchange. Names a host that processes incoming mail for the designated target. The target may contain wildcards such as *.ATT.COM, so that a single MX record can redirect the mail for an entire subtree.
CNAME	An alias for the real name of the host
PTR	Used to map IP addresses to host names
HINFO	Host type and operating system information. This can supply a hacker with a list of targets susceptible to a particular operating system weakness. This record is rare, and that is good.
WKS	Well-known services, a list of supported protocols. It is rarely used, but could save an attacker an embarrassing port scan.
SRV	Service Location — use the DNS to find out how to get to contact a particular service. Also see NAPTR.
SIG	A signature record; used as part of DNSsec
DNSKEY	A public key for DNSsec
NAPTR	Naming Authority Pointer, for indirection

the individual hops will change frequently, the so-called AS path to nearby, major destinations is likely to remain relatively stable. The *traceroute-as* package can help with this.

2.2.2   The Domain Name System

The *Domain Name System (DNS)* [Mockapetris, 1987a, 1987b; Lottor, 1987; Stahl, 1987] is a distributed database system used to map host names to IP addresses, and vice versa. (Some vendors call DNS *bind*, after a common implementation of it [Albitz and Liu, 2001].) In its normal mode of operation, hosts send UDP queries to DNS servers. Servers reply with either the proper answer or information about smarter servers. Queries can also be made via TCP, but TCP operation is usually reserved for *zone transfers*. Zone transfers are used by backup servers to obtain a full copy of their portion of the namespace. They are also used by hackers to obtain a list of targets quickly.

A number of different sorts of *resource records (RRs)* are stored by the DNS. An abbreviated list is shown in Table 2.1.

The DNS namespace is tree structured. For ease of operation, subtrees can be delegated to other servers. Two logically distinct trees are used. The first tree maps host names such as

SMTP.ATT.COM to addresses like 192.20.225.4. Other per-host information may optionally be included, such as HINFO or MX records. The second tree is for *inverse queries*, and contains PTR records. In this case, it would map 4.225.20.192.IN-ADDR.ARPA to SMTP.ATT.COM. There is no enforced relationship between the two trees, though some sites have attempted to mandate such a link for some services. The inverse tree is seldom as well-maintained and up-to-date as the commonly used forward mapping tree.

There are proposals for other trees, but they are not yet widely used.

9   The separation between forward naming and backward naming can lead to trouble. A hacker who controls a portion of the inverse mapping tree can make it lie. That is, the inverse record could falsely contain the name of a machine your machine trusts. The attacker then attempts an *rlogin* to your machine, which, believing the phony record, will accept the call.

Most newer systems are now immune to this attack. After retrieving the putative host name via the DNS, they use that name to obtain their set of IP addresses. If the actual address used for the connection is not in this list, the call is bounced and a security violation logged.

The cross-check can be implemented in either the library subroutine that generates host names from addresses (gethostbyaddr on many systems) or in the daemons that are extending trust based on host name. It is important to know how your operating system does the check; if you do not know, you cannot safely replace certain pieces. Regardless, whichever component detects an anomaly should log it.

10   There is a more damaging variant of this attack [Bellovin, 1995]. In this version, the attacker contaminates the target's cache of DNS responses prior to initiating the call. When the target does the cross-check, it appears to succeed, and the intruder gains access. A variation on this attack involves flooding the target's DNS server with phony responses, thereby confusing it. We've seen hacker's toolkits with simple programs for poisoning DNS caches.

Although the very latest implementations of the DNS software seem to be immune to this, it is imprudent to assume that there are no more holes. We strongly recommend that exposed machines not rely on name-based authentication. Address-based authentication, though weak, is far better.

There is also a danger in a feature available in many implementations of DNS resolvers [Gavron, 1993]. They allow users to omit trailing levels if the desired name and the user's name have components in common. This is a popular feature: Users generally don't like to spell out the fully qualified domain name.

For example, suppose someone on SQUEAMISH.CS.BIG.EDU tries to connect to some destination FOO.COM. The resolver would try FOO.COM.CS.BIG.EDU, FOO.COM.BIG.EDU, and FOO.COM.EDU before trying (the correct) FOO.COM. Therein lies the risk. If someone were to create a domain COM.EDU, they could intercept traffic intended for anything under .COM. Furthermore, if they had any wildcard DNS records, the situation would be even worse. A cautious user may wish to use a *rooted domain name*, which has a trailing period. In this example, the resolver won't play these games for the address X.CS.BIG.EDU. (note the trailing period). A cautious system administrator should set the search sequence so that only the local domain is checked for unqualified names.

Authentication problems aside, the DNS is problematic for other reasons. It contains a wealth of information about a site: Machine names and addresses, organizational structure, and so on.

Think of the joy a spy would feel on learning of a machine named FOO.7ESS.MYMEGACORP.COM, and then being able to dump the entire 7ESS.MYMEGACORP.COM domain to learn how many computers were allocated to developing a new telephone switch.

Some have pointed out that people don't put their secrets in host names, and this is true. Names analysis can provide useful information, however, just as traffic analysis of undeciphered messages can be useful.

Keeping this information from the overly curious is hard. Restricting zone transfers to the authorized secondary servers is a good start, but clever attackers can exhaustively search your network address space via DNS inverse queries, giving them a list of host names. From there, they can do forward lookups and retrieve other useful information. Furthermore, names leak in other ways, such as Received: lines in mail messages. It's worth some effort to block such things, but it's probably not worth too much effort or too much worry; names will leak, but the damage isn't great.

DNSsec

The obvious way to fix the problem of spoofed DNS records is to digitally sign them. Note, though, that this doesn't eliminate the problem of the inverse tree—if the owner of a zone is corrupt, he or she can cheerfully sign a fraudulent record. This is prevented via a mechanism known as *DNSsec* [Eastlake, 1999]. The basic idea is simple enough: All "RRsets" in a secure zone have a SIG record. Public keys (signed, of course) are in the DNS tree, too, taking the place of certificates. Moreover, a zone can be signed offline, thereby reducing the exposure of private zone-signing keys.

As always, the devil is in the details. The original versions [Eastlake and Kaufman, 1997; Eastlake, 1999] were not operationally sound, and the protocol was changed in incompatible ways. Other issues include the size of signed DNS responses (DNS packets are limited to 512 bytes if sent by UDP, though this is addressed by EDNS0 [Vixie, 1999]); the difficulty of signing a massive zone like .COM; how to handle DNS dynamic update; and subtleties surrounding wildcard DNS records. There's also quite a debate going on about "opt-in": Should it be possible to have a zone (such as .COM) where only some of the names are signed?

These issues and more have delayed any widespread use of DNSsec. At this time, it appears likely that deployment will finally start in 2003, but we've been overly optimistic before.

2.2.3   BOOTP and DHCP

The *Dynamic Host Configuration Protocol (DHCP)* is used to assign IP addresses and supply other information to booting computers (or ones that wake up on a new network). The booting client emits UDP broadcast packets and a server replies to the queries. Queries can be forwarded to other networks using a relay program. The server may supply a fixed IP address, usually based on the Ethernet address of the booting host, or it may assign an address out of a pool of available addresses. DHCP is an extension of the older, simpler BOOTP protocol. Whereas BOOTP only delivers a single message at boot time, DHCP extensions provide for updates or changes to IP addresses and other information after booting. DHCP servers often interface with a DNS server

to provide current IP/name mapping. An authentication scheme has been devised [Droms and Arbaugh, 2001], but it is rarely used.

The protocol can supply quite a lot of information—the domain name server and default route address and the default domain name as well as the client's IP address. Most implementations will use this information. It can also supply addresses for things such as the network time service, which is ignored by most implementations.

For installations of any size, it is nearly essential to run DHCP. It centralizes the administration of IP addresses, simplifying administrative tasks. Dynamic IP assignments conserve scarce IP address space usage. It easily provides IP addresses for visiting laptop computers—coffeeshops that provide wireless Internet access have to run this protocol. DHCP relay agents eliminate the need for a DHCP server on every LAN segment.

DHCP logs are important for forensics, especially when IP addresses are assigned dynamically. It is often important to know which hardware was associated with an IP address at a given time; the logged Ethernet address can be very useful. Law enforcement is often very interested in ISP DHCP logs (and RADIUS or other authentication logs; see Section 7.7) shortly after a crime is detected.

The protocol is used on local networks, which limits the security concerns somewhat. Booting clients broadcast queries to the local network. These can be forwarded elsewhere, but either the server or the relay agent needs access to the local network. Because the booting host doesn't know its own IP address yet, the response must be delivered to its layer 2 address, usually its Ethernet address. The server does this by either adding an entry to its own ARP table or emitting a raw layer 2 packet. In any case, this requires direct access to the local network, which a remote attacker doesn't have.

Because the DHCP queries are generally unauthenticated, the responses are subject to man-in-the-middle and DOS attacks, but if an attacker already has access to the local network, then he or she can already perform ARP-spoofing attacks (see Section 2.1.2). That means there is little added risk in choosing to run the BOOTP/DHCP protocol. The interface with the DNS server requires a secure connection to the DNS server; this is generally done via the symmetric-key variant of `SIG` records.

Rogue DHCP servers can beat the official server to supplying an answer, allowing various attacks. Or, they can swamp the official server with requests from different simulated Ethernet addresses, consuming all the available IP addresses.

Finally, some DHCP clients implement lease processing dangerously. For example, *dhclient*, which runs on many UNIX systems, leaves a UDP socket open, with a privileged client program, running for the duration. This is an unnecessary door into the client host: It need only be open for occasional protocol exchanges.

2.3   IP version 6

IP version 6 (IPv6) [Deering and Hinden, 1998] is much like the current version of IP, only more so. The basic philosophy—IP is an unreliable datagram protocol, with a minimal header—is the

same, but there are approximately $\aleph_0$ details that matter. Virtually all of the supporting elements are more complex.

11 The most important thing to know about IPv6 is that easy renumbering is one of the design goals. This means that any address-based access controls need to know about renumbering, and need to be updated at the right times. Of course, they need to know about *authentic* renumbering events; fraudulent ones should, of course, be treated with the proper mix of disdain and contempt.

Renumbering doesn't occur instantaneously throughout a network. Rather, the new prefix—the low-order bits of hosts addresses are not touched during renumbering—is phased in gradually. At any time, any given interface may have several addresses, with some labeled "deprecated," i.e., their use is discouraged for new connections. Old connections, however, can continue to use them for quite some time, which means that firewalls and the like need to accept them for a while, too.

2.3.1   IPv6 Address Formats

IPv6 addresses aren't simple 128-bit numbers. Rather, they have structure [Hinden and Deering, 1998], and the structure has semantic implications. There are many different forms of address, and any interface can have many separate addresses of each type simultaneously.

12 The simplest address type is the *global unicast address*, which is similar to IPv4 addresses. In the absence of other configuration mechanisms, such as a DHCP server or static addresses, hosts can generate their own IPv6 address from the local prefix (see Section 2.3.2) and their MAC address. Because MAC addresses tend to be constant for long periods of time, a mechanism is defined to create temporary addresses [Narten and Draves, 2001]. This doesn't cause much trouble for firewalls, unless they're extending trust on the basis of source addresses (i.e., if they're misconfigured). But it does make it a lot harder to track down a miscreant's machine after the fact. If you need to do that, your routers will need to log what MAC addresses are associated with what IPv6 addresses—and routers are not, in general, designed to log such things.

There is a special subset of unicast addresses known as *anycast addresses*. Many different nodes may share the same anycast address; the intent is that clients wishing to connect to a server at such an address will find the closest instance of it. "Close" is measured "as the packets fly," i.e., the instance that the routing system thinks is closest.

Another address type is the *site-local address*. Site-local addresses are used within a "site"; border routers are supposed to ensure that packets containing such source or destination addresses do not cross the boundary. This might be a useful security property *if* you are sure that your border routers enforce this properly.

At press time, there was no consensus on what constitutes a "site." It is reasonably likely that the definition will be restricted, especially compared to the (deliberate) early vagueness. In particular, a site is likely to have a localized view of the DNS, so that one player's internal addresses aren't visible to others. Direct routing between two independent sites is likely to be banned, too, so that routers don't have to deal with two or more different instances of the same address.

It isn't at all clear that a site boundary is an appropriate mechanism for setting security policy. If nothing else, it may be too large. Worse yet, such a mechanism offers no opportunity for finer-grained access controls.

Link-local addresses are more straightforward. They can only be used on a single link, and are never forwarded by routers. Link-local addresses are primarily used to talk to the local router, or during address configuration.

Multicast is a one-to-many mechanism that can be thought of as a subset of broadcast. It is a way for a sender to transmit an IP packet to a group of hosts. IPv6 makes extensive use of multicast; things that were done with broadcast messages in IPv4, such as routing protocol exchanges, are done with multicast in IPv6. Thus, the address FF02:0:0:0:0:0:0:2 means "all IPv6 routers on this link." Multicast addresses are scoped; there are separate classes of addresses for nodes, links, sites, and organizations, as well as the entire Internet. Border routers must be configured properly to avoid leaking confidential information, such as internal videocasts.

2.3.2   Neighbor Discovery

In IPv6, ARP is replaced by the *Neighbor Discovery (ND)* protocol [Narten *et al.*, 1998]. ND is much more powerful, and is used to set many parameters on end systems. This, of course, means that abuse of ND is a serious matter; unfortunately, at the moment there are no well-defined mechanisms to secure it. (The ND specification speaks vaguely of using *Authentication Header (AH)* (which is part of IPsec), but doesn't explain how the relevant security associations should be set up.) There is one saving grace: ND packets *must* have their hop limit set to 255, which prevents off-link nodes from sending such packets to an unsuspecting destination.

Perhaps the most important extra function provided by ND is prefix announcement. Routers on a link periodically multicast *Router Advertisement (RA)* messages; hosts receiving such messages update their prefix lists accordingly. RA messages also tell hosts about routers on their link; false RA messages are a lovely way to divert traffic.

The messages are copiously larded with timers: what the lifetime of a prefix is, how long a default route is good for, the time interval between retransmissions of *Neighbor Solicitation* messages, and so on.

2.3.3   DHCPv6

Because one way of doing something isn't enough, IPv6 hosts can also acquire addresses via IPv6's version of DHCP. Notable differences from IPv4's DHCP include the capability to assign multiple addresses to an interface, strong bidirectional authentication, and an optional mechanism for revocation of addresses before their leases expire. The latter mechanism requires clients to listen continually on their DHCP ports, which may present a security hazard; no other standards mandate that client-only machines listen on any ports. On the other hand, the ability to revoke leases can be very useful if you've accidentally set the lease time too high, or if you want to bring down a DHCP server for emergency maintenance during lease lifetime. Fortunately, this feature is supposed to be configurable; we suggest turning it off, and using modest lease times instead.

2.3.4   Filtering IPv6

We do not have wide area IPv6 yet on most of the planet, so several protocols have been developed to carry IPv6 over IPv4. If you do not want IPv6, tunneled traffic should be blocked. If you want

IPv6 traffic (and you're reading this book), you'll need an IPv6 firewall. If your primary firewall doesn't do this, you'll need to permit IPv6 tunnels, but only if they terminate on the outside of your IPv6 firewall. This needs to be engineered with caution.

There are several ways to tunnel IPv6 over an IPv4 cloud. RFC 3056 [Carpenter and Moore, 2001] specifies a protocol called *6to4*, which encapsulates v6 traffic in IPv4 packets with the protocol number 41. There is running code for *6to4* in the various BSD operating systems. Another protocol, *6over4* [Carpenter and Jung, 1999], is similar. Packet filters can recognize this traffic and either drop it or forward it to something that knows what to do with tunneled traffic. The firewall package *ipf*, discussed in Section 11.3.2, can filter IPv6; however, many current firewalls do not.

Another scheme for tunneling IPv6 over IPv4 is called *Teredo*. (*Teredo navalis* is a shipworm that bores its way through wooden structures and causes extensive damage to ships and other wooden structures.) The protocol uses UDP port 3544 and permits tunneling through *Network Address Translation (NAT)* boxes [Srisuresh and Egevang, 2001]. If you are concerned about this, block UDP port 3544. While it is always prudent to block all UDP ports, except the ones that you explicitly want to open, it is especially important to make sure that firewalls block this one. If used from behind a NAT box, Teredo relies on an outside server with a globally routable address. Given the difficulty of knowing how many NAT boxes one is behind, especially as the number can vary depending on your destination, this scheme is controversial. It is not clear if or when it will be standardized.

A final scheme for tunneling IPv6 over today's Internet is based on circuit relays [Hagino and Yamamoto, 2001]. With these, a router-based relay agent maps individual IPv6 TCP connections to IPv4 TCP connections; these are converted back at the receiving router.

2.4   Network Address Translators

We're running out of IP addresses. In fact, some would say that we have already run out. The result has been the proliferation of NAT boxes [Srisuresh and Holdrege, 1999; Tsirtsis and Srisuresh, 2000; Srisuresh and Egevang, 2001]. Conceptually, NATs are simple: they listen on one interface (which probably uses so-called *private address space* [Rekhter *et al.*, 1996]), and rewrite the source address and port numbers on outbound packets to use the public source IP address assigned to the other interface. On reply packets, they perform the obvious inverse operation. But life in the real world isn't that easy.

Many applications simply won't work through NATs. The application data contains embedded IP addresses (see, for example, the description of FTP in Section 3.4.2); if the NAT doesn't know how to also rewrite the data stream, things will break.

Incoming calls to dynamic ports don't work very well either. Most NAT boxes will let you route traffic to specific static hosts and ports; they can't cope with arbitrary application protocols.

To be sure, commercial NATs do know about common higher-level protocols. But if you run something unusual, or if a new one is developed and your vendor doesn't support it (or doesn't support it on your box, if it's more than a year or so old), you're out of luck.

From a security perspective, a more serious issue is that NATs don't get along very well with encryption. Clearly, a NAT can't examine an encrypted application stream. Less obviously, some forms of IPsec (see Section 18.3) are incompatible with NAT. IPsec can protect the transport layer header, which includes a checksum; this checksum includes the IP address that the NAT box needs to rewrite. These issues and many more are discussed in [Hain, 2000; Holdrege and Srisuresh, 2001; Senie, 2002].

Some people think that NAT boxes are a form of firewall. In some sense, they are, but they're low-end ones. At best, they're a form of packet filter (see Section 9.1). They lack the application-level filtering that most dedicated firewalls have; more importantly, they may lack the necessarily paranoid designers. To give just one example, some brands of home NAT boxes are managed via the Web—via an unencrypted connection only. Fortunately, you can restrict its management service to listen on the inside interface only.

We view the proliferation of NATs as an artifact of the shortage of IPv4 address space. The protocol complexities they introduce make them chancy. Use a real firewall, and hope that IPv6 comes soon.

2.5   Wireless Security

A world of danger can lurk at the link layer. We've already discussed ARP-spoofing. But wireless networks add a new dimension. It's not that they extend the attackers' powers; rather, they expand the reach and number of potential attackers.

The most common form of wireless networking is IEEE 802.11b, known to marketeers as WiFi. 802.11 is available in most research labs, at universities, at conferences, in coffeehouses, at airports, and even in peoples' homes. To prevent random, casual access to these networks, the protocol designers added a symmetric key encryption algorithm called *Wired Equivalent Privacy (WEP)*.

The idea is that every machine on the wireless network is configured with a secret key, and thus nobody without the key can eavesdrop on traffic or use the network. Although the standard supports encryption, early versions supported either no encryption at all or a weak 40-bit algorithm. As a result, you can cruise through cities or high-tech residential neighborhoods and obtain free Internet (or intranet!) access, complete with DHCP support! Mark Seiden coined the term *war driving* for this activity.

Unfortunately, the designers of 802.11 did not get the protocol exactly right. The security flaws resulted from either ignorance of or lack of attention to known techniques. A team of researchers consisting of Nikita Borisov, Ian Goldberg, and David Wagner [2001] discovered a number of flaws that result in attackers being able to do the following: decrypt traffic based on statistical analysis; inject new traffic from unauthorized mobile stations; decrypt traffic based on tricking the access points; and decrypt all traffic after passively analyzing a day's worth of traffic.

This is devastating. In most places, the 802.11 key does not change after deployment, if it is used at all. Considering the huge deployed base of 802.11 cards and access points, it will be a monumental task to fix this problem.

A number of mistakes were made in the design. Most seriously, it uses a stream cipher, which is poorly matched to the task. (See Appendix A for an explanation of these terms.) All users on a network share a common, static key. (Imagine the security of sharing that single key in a community of college students!) The alleged *initialization vector (IV)* used is 24 bits long, guaranteeing frequent collisions for busy access points. The integrity check used by WEP is a CRC-32 checksum, which is linear. In all cases, it would have been trivial to avoid trouble. They should have used a block cipher; failing that, they should have used much longer IVs and a cryptographic checksum. Borisov *et al.* [2001] implemented the passive attack.

WEP also comes with an authentication mechanism. This, too, was easily broken [Arbaugh *et al.*, 2001]. The most devastating blow to WEP, however, came from a theoretical paper that exposed weaknesses in RC4, the underlying cipher in WEP [Fluhrer *et al.*, 2001]. The attack (often referred to as the FMS attack) requires one byte of known plaintext and several million packets, and results in a passive adversary directly recovering the key. Because 802.11 packets are encapsulated in 802.2 headers with a constant first byte, all that is needed is the collection of the packets.

Within a week of the release of this paper, researchers had implemented the attack [Stubble-field *et al.*, 2002], and shortly thereafter, two public tools *Airsnort* and *WEPCrack* appeared on the Web.

13 Given the availability of these programs, WEP can be considered dead in the water. It provides a sense of security, without useful security. This is worse than providing no security at all because some people will trust it. Our recommendation is to put your wireless network outside your firewall, turn on WEP as another, almost useless security layer, and use remote access technology such as an IPsec VPN or *ssh* to get inside from the wireless network.

14 Remember that just because you cannot access your wireless network with a PCMCIA card from the parking lot, it does not mean that someone with an inexpensive high gain antenna cannot reach it from a mile (or twenty miles!) away. In fact, we have demonstrated that a standard access point inside a building is easily reachable from that distance.

On the other hand, you cannot easily say "no" to insiders who want wireless convenience. Access points cost under $150; beware of users who buy their own and plug them into the wall jacks of your internal networks. Periodic scanning for rogue access points is a must. (Nor can you simply look for the MAC address of authorized hosts; many of the commercial access points come with a MAC address cloning feature.)

2.5.1   Fixing WEP

Given the need to improve WEP before all of the hardware is redesigned and redeployed in new wireless cards, the IEEE came up with a replacement called *Temporal Key Integrity Protocol (TKIP)*. TKIP uses the existing API on the card—namely, RC4 with publicly visible IVs—and plays around with the keys so that packets are dynamically keyed. In TKIP, keys are changed often (on the order of hours), and IVs are forced to change with no opportunity to wrap around. Also, the checksum on packets is a cryptographic MAC, rather than the CRC used by WEP. Thus, TKIP is not vulnerable to the Berkeley attacks, nor to the FMS one. It is a reasonable workaround, given

the legacy issues involved. The next generation of hardware is designed to support the *Advanced Encryption Standard (AES)*, and is being scrutinized by the security community.

It is not clear that the link layer is the right one for security. In a coffeeshop, the security association is terminated by the store: is there any reason you should trust the shopkeeper? Perhaps link-layer security makes some sense in a home, where you control both the access point and the wireless machines. However, we prefer end-to-end security at the network layer or in the applications.