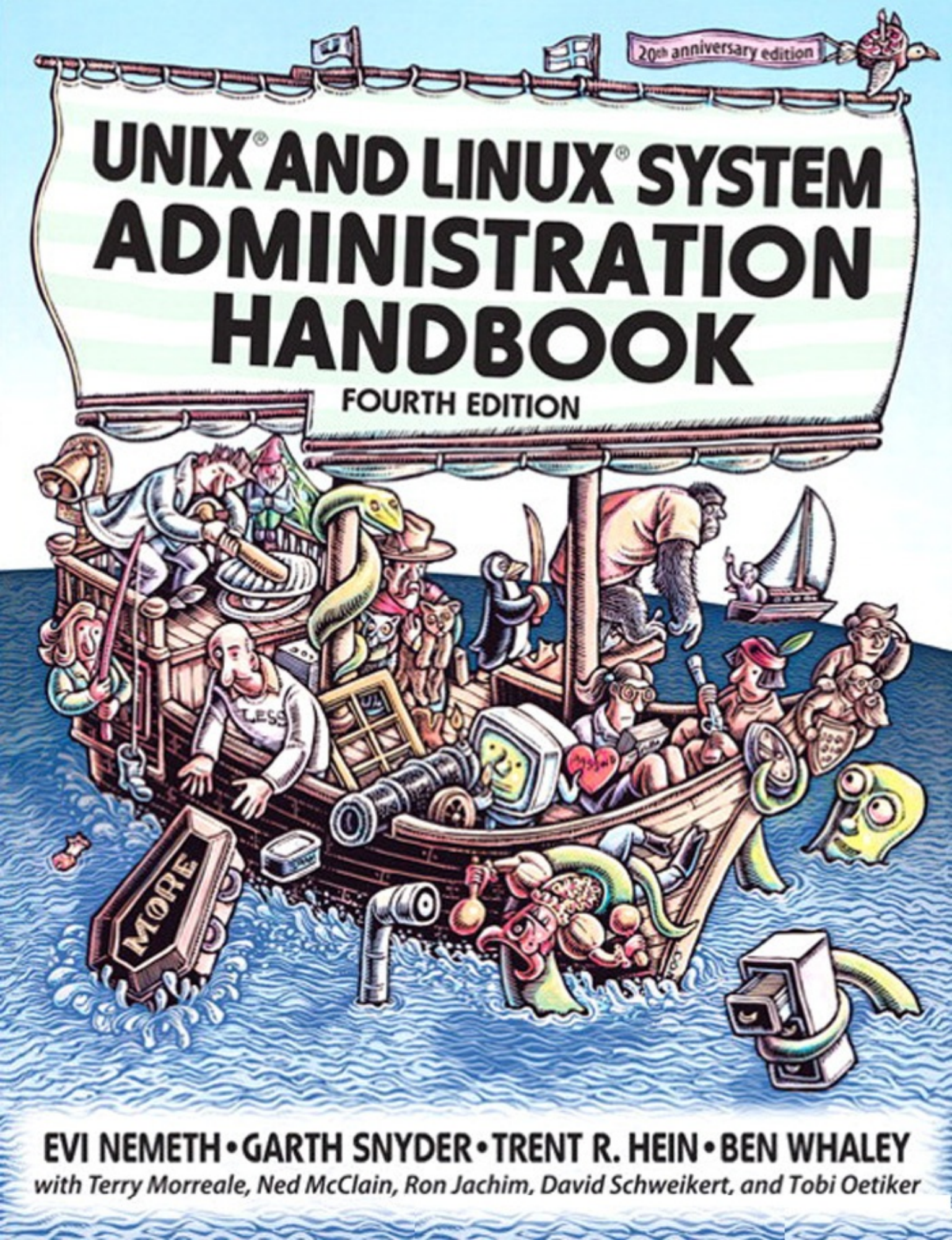
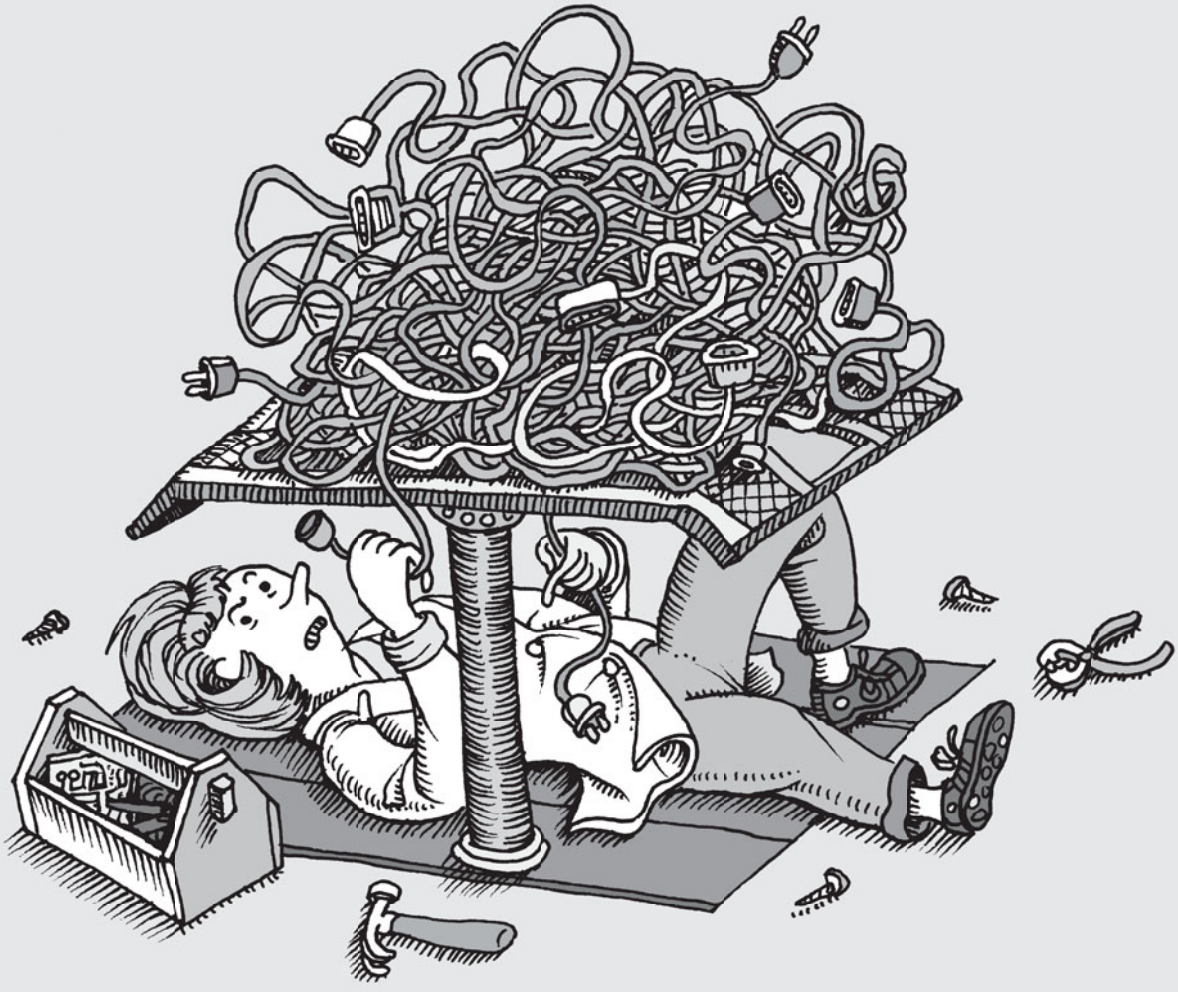

20th anniversary edition
UNIX® AND LINUX® SYSTEM ADMINISTRATION HANDBOOK
FOURTH EDITION
LESS
MORE
EVI NEMETH • GARTH SNYDER • TRENT R. HEIN • BEN WHALEY
with Terry Morreale, Ned McClain, Ron Jachim, David Schweikert, and Tobi Oetiker

SECTION TWO

NETWORKING



14 TCP/IP Networking

It would be hard to overstate the importance of networks to modern computing, although that doesn't seem to stop people from trying. At many sites—perhaps even the majority—web and email access are the primary uses of computers. As of 2010, internetworldstats.com estimates the Internet to have nearly 1.5 billion users, or more than 21% of the world's population. In North America, Internet penetration approaches 75%.

TCP/IP is the networking system that underlies the Internet. TCP/IP does not depend on any particular hardware or operating system, so devices that speak TCP/IP can all exchange data ("interoperate") despite their many differences.

TCP/IP works on networks of any size or topology, whether or not they are connected to the outside world. This chapter introduces the TCP/IP protocols in the political and technical context of the Internet, but stand-alone networks are quite similar at the TCP/IP level.

14.1 TCP/IP AND ITS RELATIONSHIP TO THE INTERNET

TCP/IP and the Internet share a history that goes back several decades. The technical success of the Internet is due largely to the elegant and flexible design of

TCP/IP and to the fact that TCP/IP is an open and nonproprietary protocol suite. In turn, the leverage provided by the Internet has helped TCP/IP prevail over several competing protocol suites that were favored at one time or another for political or commercial reasons.

The progenitor of the modern Internet was a research network called ARPANET established in 1969 by the U.S. Department of Defense. By the end of the 1980s the network was no longer a research project and we transitioned to the commercial Internet. Today's Internet is a collection of private networks owned by Internet service providers (ISPs) that interconnect at many so-called peering points.

Who runs the Internet?

Oversight of the Internet and the Internet protocols has long been a cooperative and open effort, but its exact structure has changed as the Internet has evolved into a public utility and a driving force in the world economy. Current Internet governance is split roughly into administrative, technical, and political wings, but the boundaries between these functions are often vague. The major players are listed below:

- ICANN, the Internet Corporation for Assigned Names and Numbers: if any one group can be said to be in charge of the Internet, this is probably it. It's the only group with any sort of actual enforcement capability. ICANN controls the allocation of Internet addresses and domain names, along with various other snippets such as protocol port numbers. It is organized as a nonprofit corporation headquartered in California and operates under a memorandum of understanding with the U.S. Department of Commerce. (icann.org)
- ISOC, the Internet Society: ISOC is an open-membership organization that represents Internet users. Although it has educational and policy functions, it's best known as the umbrella organization for the technical development of the Internet. In particular, it is the parent organization of the Internet Engineering Task Force (ietf.org), which oversees most technical work. ISOC is an international nonprofit organization with offices in Washington, D.C. and Geneva. (isoc.org)
- IGF, the Internet Governance Forum: a relative newcomer, the IGF was created by the United Nations in 2005 to establish a home for international and policy-oriented discussions related to the Internet. It's currently structured as a yearly conference series, but its importance is likely to grow over time as governments attempt to exert more control over the operation of the Internet. (intgovforum.org)

Of these groups, ICANN has the toughest job: establishing itself as the authority in charge of the Internet, undoing the mistakes of the past, and foreseeing the future, all while keeping users, governments, and business interests happy.

Network standards and documentation

If your eyes haven't glazed over just from reading the title of this section, you've probably already had several cups of coffee. Nonetheless, accessing the Internet's authoritative technical documentation is a crucial skill for system administrators, and it's more entertaining than it sounds.

The technical activities of the Internet community are summarized in documents known as Requests for Comments or RFCs. Protocol standards, proposed changes, and informational bulletins all usually end up as RFCs. RFCs start their lives as Internet Drafts, and after lots of email wrangling and IETF meetings they either die or are promoted to the RFC series. Anyone who has comments on a draft or proposed RFC is encouraged to reply. In addition to standardizing the Internet protocols, the RFC mechanism sometimes just documents or explains aspects of existing practice.

RFCs are numbered sequentially; currently, there are about 5,600. RFCs also have descriptive titles (e.g., *Algorithms for Synchronizing Network Clocks*), but to forestall ambiguity they are usually cited by number. Once distributed, the contents of an RFC are never changed. Updates are distributed as new RFCs with their own reference numbers. Updates may either extend and clarify existing RFCs or supersede them entirely.

RFCs are available from numerous sources, but rfc-editor.org is dispatch central and will always have the most up-to-date information. Look up the status of an RFC at rfc-editor.org before investing the time to read it; it may no longer be the most current document on that subject.

The Internet standards process itself is detailed in RFC2026. Another useful meta-RFC is RFC5540, *40 Years of RFCs*, which describes some of the cultural and technical context of the RFC system.

Don't be scared away by the wealth of technical detail found in RFCs. Most contain introductions, summaries, and rationales that are useful for system administrators even when the technical details are not. Some RFCs are specifically written as overviews or general introductions. RFCs may not be the gentlest way to learn about a topic, but they are authoritative, concise, and free.

Not all RFCs are full of boring technical details. Here are some of our favorites on the lighter side (usually written on April 1st):

- RFC1149 – *Standard for Transmission of IP Datagrams on Avian Carriers*[1]
- RFC1925 – *The Twelve Networking Truths*
- RFC3251 – *Electricity over IP*
- RFC3514 – *The Security Flag in the IPv4 Header*
- RFC4041 – *Requirements for Morality Sections in Routing Area Drafts*

1. A group of Linux enthusiasts from BLUG, the Bergen (Norway) Linux User Group, actually implemented the Carrier Pigeon Internet Protocol (CPIP) as specified in RFC1149. For details, see the web site blug.linux.no/rfc1149.

In addition to being assigned its own serial number, an RFC can also be assigned an FYI (For Your Information) number, a BCP (Best Current Practice) number, or a STD (Standard) number. FYIs, STDs, and BCPs are subseries of the RFCs that include documents of special interest or importance.

FYIs are introductory or informational documents intended for a broad audience. They can be a good place to start research on an unfamiliar topic if you can find one that's relevant. Unfortunately, this series has languished recently and not many of the FYIs are up to date.

BCPs document recommended procedures for Internet sites; they consist of administrative suggestions and for system administrators are often the most valuable of the RFC subseries.

STDs document Internet protocols that have completed the IETF's review and testing process and have been formally adopted as standards.

RFCs, FYIs, BCPs, and STDs are numbered sequentially within their own series, so a document can bear several different identifying numbers. For example, RFC1713, *Tools for DNS Debugging*, is also known as FYI27.

14.2 NETWORKING ROAD MAP

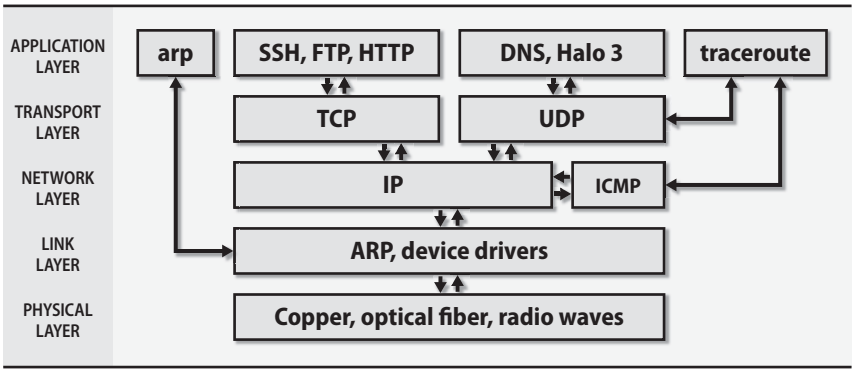
Now that we've provided a bit of context, let's look at the TCP/IP protocols themselves. TCP/IP is a protocol "suite," a set of network protocols designed to work smoothly together. It includes several components, each defined by a standards-track RFC or series of RFCs:

- IP, the Internet Protocol, which routes data packets from one machine to another (RFC791)
- ICMP, the Internet Control Message Protocol, which provides several kinds of low-level support for IP, including error messages, routing assistance, and debugging help (RFC792)
- ARP, the Address Resolution Protocol, which translates IP addresses to hardware addresses (RFC826)[2]
- UDP, the User Datagram Protocol, which provides unverified, one-way data delivery (RFC768)
- TCP, the Transmission Control Protocol, which implements reliable, full duplex, flow-controlled, error-corrected conversations (RFC793)

These protocols are arranged in a hierarchy or "stack", with the higher-level protocols making use of the protocols beneath them. TCP/IP is conventionally described as a five-layer system (as shown in Exhibit A), but the actual TCP/IP protocols inhabit only three of these layers.

2. This is actually a little white lie. ARP is not really part of TCP/IP and can be used with other protocol suites. However, it's an integral part of the way TCP/IP works on most LAN media.

Exhibit A TCP/IP layering model



IPv4 and IPv6

The version of TCP/IP that has been in widespread use for three decades is protocol revision 4, aka IPv4. It uses four-byte IP addresses. A modernized version, IPv6, expands the IP address space to 16 bytes and incorporates several other lessons learned from the use of IPv4. It removes several features of IP that experience has shown to be of little value, making the protocol potentially faster and easier to implement. IPv6 also integrates security and authentication into the basic protocol.

All modern operating systems and many network devices already support IPv6. However, active use of IPv6 remains essentially zero in the real world.[3] Experience suggests that it's probably best for administrators to defer production use of IPv6 to the extent that this is possible. Everyone will eventually be forced to switch to IPv6, but as of 2010 that day is still years away. At the same time, the transition is not so far in the future that you can ignore it when purchasing new network devices. Insist on IPv6 compatibility for new acquisitions.

The development of IPv6 was to a large extent motivated by the concern that we are running out of 4-byte IPv4 address space. And indeed we are: projections indicate that the current IPv4 allocation system will collapse some time around 2011. (See ipv4.potaroo.net for a daily update.) Even so, mainstream adoption of IPv6 throughout the Internet is probably still not in the cards anytime soon.

More likely, another round of stopgap measures on the part of ISPs and ICANN (or more specifically, its subsidiary IANA, the Internet Assigned Numbers Authority) will extend the dominance of IPv4 for another few years. We expect to see wider use of IPv6 on the Internet backbone, but outside of large ISPs, academic sites involved in Internet research, and universal providers such as Google, our

3. A Google study presented at RIPE 57 in October 2008 indicated that overall IPv6 penetration (actual use, not capability) was 0.24%. No country had IPv6 penetration greater than 0.76%.

guess is that IPv6 will not be directly affecting most sysadmins' work in the immediate future.

The IPv4 address shortage is felt more acutely outside the United States, and so IPv6 has received a warmer welcome there. In the United States, it may take a killer application to boost IPv6 over the hill: for example, a new generation of cell phones that map an IPv6 address to a telephone number. (Voice-over-IP systems would also benefit from a closer correspondence between phone numbers and IPv6 addresses.)

In this book, we focus on IPv4 as the mainstream version of TCP/IP. IPv6-specific material is explicitly marked. Fortunately for sysadmins, IPv4 and IPv6 are highly analogous. If you understand IPv4, you already know most of what you need to know about IPv6. The main difference between the versions lies in their addressing schemes. In addition to longer addresses, IPv6 introduces a few additional addressing concepts and some new notation. But that's about it.

Packets and encapsulation

TCP/IP supports a variety of physical networks and transport systems, including Ethernet, token ring, MPLS (Multiprotocol Label Switching), wireless Ethernet, and serial-line-based systems. Hardware is managed within the link layer of the TCP/IP architecture, and higher-level protocols do not know or care about the specific hardware being used.

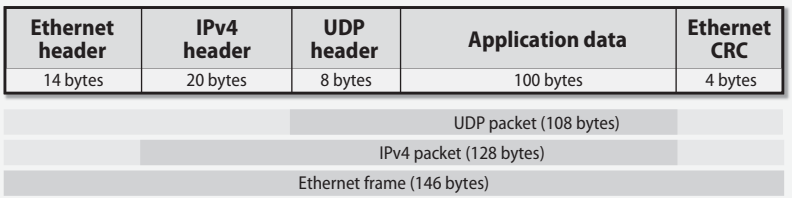
Data travels on a network in the form of *packets*, bursts of data with a maximum length imposed by the link layer. Each packet consists of a header and a payload. The header tells where the packet came from and where it's going. It can also include checksums, protocol-specific information, or other handling instructions. The payload is the data to be transferred.

The name of the primitive data unit depends on the layer of the protocol. At the link layer it is called a *frame*, at the IP layer a *packet*, and at the TCP layer a *segment*. In this book, we use "packet" as a generic term that encompasses these various cases.

As a packet travels down the protocol stack (from TCP or UDP transport to IP to Ethernet to the physical wire) in preparation for being sent, each protocol adds its own header information. Each protocol's finished packet becomes the payload part of the packet generated by the next protocol. This nesting is known as encapsulation. On the receiving machine, the encapsulation is reversed as the packet travels back up the protocol stack.

For example, a UDP packet being transmitted over Ethernet contains three different wrappers or envelopes. On the Ethernet wire, it is framed with a simple header that lists the source and next-hop destination hardware addresses, the length of the frame, and the frame's checksum (CRC). The Ethernet frame's payload is an IP packet, the IP packet's payload is a UDP packet, and the UDP packet's payload is the data being transmitted. Exhibit B shows the components of such a frame.

Exhibit B A typical network packet[4]



Ethernet framing

One of the main chores of the link layer is to add headers to packets and to put separators between them. The headers contain each packet's link-layer addressing information and checksums, and the separators ensure that receivers can tell where one packet stops and the next one begins. The process of adding these extra bits is known generically as framing.

The link layer is actually divided into two parts: MAC, the Media Access Control sublayer, and LLC, the Link Layer Control sublayer. The MAC layer deals with the media and transmits packets onto the wire. The LLC layer handles the framing.

Today, a single standard for Ethernet framing is in common use: DIX Ethernet II. Historically, several slightly different standards based on IEEE 802.2 were also used, especially on Novell networks.

Maximum transfer unit

The size of packets on a network may be limited both by hardware specifications and by protocol conventions. For example, the payload of a standard Ethernet frame is traditionally 1,500 bytes. The size limit is associated with the link-layer protocol and is called the maximum transfer unit or MTU. Table 14.1 shows some typical values for the MTU.

Table 14.1 MTUs for various types of network

Network type	Maximum transfer unit
Ethernet	1,500 bytes (1,492 with 802.2 framing)[a]
FDDI	4,470 bytes (4,352 for IP/FDDI)
Token ring	Configurable[b]
PPP modem link	Configurable, often 512 or 576 bytes
Point-to-point WAN links (T1, T3)	Configurable, often 1,500 or 4,500 bytes

a. See page 541 for some comments on "jumbo" Ethernet packets.
b. Common values are 552; 1,064; 2,088; 4,508; and 8,232. Sometimes 1,500 to match Ethernet.

4. For specificity, RFCs that describe protocols often use the term "octet" instead of "byte."

The IP layer splits packets to conform to the MTU of a particular network link. If a packet is routed through several networks, one of the intermediate networks may have a smaller MTU than the network of origin. In this case, an IPv4 router that forwards the packet onto the small-MTU network further subdivides the packet in a process called fragmentation.

Fragmentation of in-flight packets is an unwelcome chore for a busy router, so IPv6 largely removes this feature. Packets can still be fragmented, but the originating host must do the work itself.

Senders can discover the lowest-MTU link through which a packet must pass by setting the packet's "do not fragment" flag. If the packet reaches an intermediate router that cannot forward the packet without fragmenting it, the router returns an ICMP error message to the sender. The ICMP packet includes the MTU of the network that's demanding smaller packets, and this MTU then becomes the governing packet size for communication with that destination.

The TCP protocol does path MTU discovery automatically, even in IPv4. UDP is not so nice and is happy to shunt extra work to the IP layer.

Fragmentation problems can be insidious. Although path MTU discovery should automatically resolve MTU conflicts, an administrator must occasionally intervene. If you are using a tunneled architecture for a virtual private network, for example, you should look at the size of the packets that are traversing the tunnel. They are often 1,500 bytes to start with, but once the tunneling header is added, they become 1,540 bytes or so and must be fragmented. Setting the MTU of the link to a smaller value averts fragmentation and increases the overall performance of the tunneled network. Consult the **ifconfig** man page to see how to set an interface's MTU.

14.3 PACKET ADDRESSING

Like letters or email messages, network packets must be properly addressed in order to reach their destinations. Several addressing schemes are used in combination:

- MAC (media access control) addresses for use by hardware
- IPv4 and IPv6 network addresses for use by software
- Hostnames for use by people

Hardware (MAC) addressing

Each of a host's network interfaces usually has one link-layer MAC address that distinguishes it from other machines on the physical network, plus one or more IP addresses that identify the interface on the global Internet. This last part bears repeating: IP addresses identify *network interfaces, not machines.* (To users the distinction is irrelevant, but administrators must know the truth.)

The lowest level of addressing is dictated by network hardware. For example, Ethernet devices are assigned a unique 6-byte hardware address at the time of manufacture. These addresses are traditionally written as a series of 2-digit hex bytes separated by colons; for example, 00:50:8D:9A:3B:DF.

Token ring interfaces have a similar address that is also six bytes long. Some point-to-point networks (such as PPP) need no hardware addresses at all; the identity of the destination is specified as the link is established.

A 6-byte Ethernet address is divided into two parts. The first three bytes identify the manufacturer of the hardware, and the last three bytes are a unique serial number that the manufacturer assigns. Sysadmins can sometimes identify the brand of machine that is trashing a network by looking up the 3-byte identifier in a table of vendor IDs. A current vendor table is available from

iana.org/assignments/ethernet-numbers

The 3-byte codes are actually IEEE Organizationally Unique Identifiers (OUIs), so you can also look up them up directly in the IEEE's database at

standards.ieee.org/regauth/oui

Of course, the relationships among the manufacturers of chipsets, components, and systems are complex, so the vendor ID embedded in a MAC address can be misleading, too.

In theory, Ethernet hardware addresses are permanently assigned and immutable. However, many network interfaces now let you override the hardware address and set one of your own choosing. This feature can be handy if you have to replace a broken machine or network card and for some reason must use the old MAC address (e.g., all your switches filter it, or your DHCP server hands out addresses based on MAC addresses, or your MAC address is also a software license key). Spoofable MAC addresses are also helpful if you need to infiltrate a wireless network that uses MAC-based access control. But for simplicity, it's generally advisable to preserve the uniqueness of MAC addresses.

IP addressing

At the next level up from the hardware, Internet addressing (more commonly known as IP addressing) is used. IP addresses are globally unique[5] and hardware independent.

See page 468 for more information about ARP.

The mapping from IP addresses to hardware addresses is implemented at the link layer of the TCP/IP model. On networks such as Ethernet that support broadcasting (that is, networks that allow packets to be addressed to "all hosts on this

5. In general, an IP address identifies a specific and unique destination. However, several special cases muddy the water. NAT (page 462) uses one interface's IP address to handle traffic for multiple machines. IP private address spaces (page 462) are addresses that multiple sites can use at once, as long as the addresses are not visible to the Internet. Anycast addressing shares one IP address among several machines.

physical network"), senders use the ARP protocol to discover mappings without assistance from a system administrator. In IPv6, an interface's MAC address can be used as part of the IP address, making the translation between IP and hardware addressing virtually automatic.

Hostname "addressing"

See Chapter 17 for more information about DNS.

IP addresses are sequences of numbers, so they are hard for people to remember. Operating systems allow one or more hostnames to be associated with an IP address so that users can type rfc-editor.org instead of 128.9.160.27. Under UNIX and Linux, this mapping can be set up in several ways, ranging from a static file (**/etc/hosts**) to the LDAP database system to DNS, the world-wide Domain Name System. Keep in mind that hostnames are really just a convenient shorthand for IP addresses, and as such, they refer to network interfaces rather than computers.

Ports

IP addresses identify a machine's network interfaces, but they are not specific enough to address individual processes or services, many of which may be actively using the network at once. TCP and UDP extend IP addresses with a concept known as a port, a 16-bit number that supplements an IP address to specify a particular communication channel. Standard services such as email, FTP, and HTTP associate themselves with "well known" ports defined in **/etc/services**.[6] To help prevent impersonation of these services, UNIX systems restrict server programs from binding to port numbers under 1,024 unless they are run as root. (Anyone can communicate with a server running on a low port number; the restriction applies only to the program listening on the port.)

Address types

The IP layer defines several broad types of address, some of which have direct counterparts at the link layer:

- Unicast – addresses that refer to a single network interface
- Multicast – addresses that simultaneously target a group of hosts
- Broadcast – addresses that include all hosts on the local subnet
- Anycast – addresses that resolve to any one of a group of hosts

Multicast addressing facilitates applications such as video conferencing in which the same set of packets must be sent to all participants. The Internet Group Management Protocol (IGMP) constructs and manages sets of hosts that are treated as one multicast destination.

Multicast is largely unused on today's Internet, but it's slightly more mainstream in IPv6. IPv6 broadcast addresses are really just specialized forms of multicast addressing.

6. You can find a full list of assigned ports at iana.org/assignments/port-numbers.

Anycast addresses bring load balancing to the network layer by allowing packets to be delivered to whichever of several destinations is closest in terms of network routing. You might expect that they'd be implemented similarly to multicast addresses, but in fact they are more like unicast addresses.

Most of the implementation details for anycast support are handled at the level of routing rather than IP. The novelty of anycast addressing is really just the relaxation of the traditional requirement that IP addresses identify unique destinations. Anycast addressing is formally described for IPv6, but the same tricks can be applied to IPv4, too—for example, as is done for root DNS name servers.

14.4 IP ADDRESSES: THE GORY DETAILS

With the exception of multicast addresses, Internet addresses consist of a network portion and a host portion. The network portion identifies a logical network to which the address refers, and the host portion identifies a node on that network. In IPv4, addresses are four bytes long and the boundary between network and host portions is set administratively. In IPv6, addresses are 16 bytes long and the network portion and host portion are always eight bytes each.

IPv4 addresses are written as decimal numbers, one for each byte, separated by periods; for example, 209.85.171.147. The leftmost byte is the most significant and is always part of the network portion.

When 127 is the first byte of an address, it denotes the "loopback network," a fictitious network that has no real hardware interface and only one host. The loopback address 127.0.0.1 always refers to the current host. Its symbolic name is "localhost". (This is another small violation of IP address uniqueness since every host thinks 127.0.0.1 is a different computer: itself.)

IPv6 addresses and their text-formatted equivalents are a bit more complicated. They're discussed in the section *IPv6 addressing* starting on page 464.

An interface's IP address and other parameters are set with the **ifconfig** command. Jump ahead to page 478 for a detailed description of **ifconfig**.

IPv4 address classes

Historically, IP addresses had an inherent "class" that depended on the first bits of the leftmost byte. The class determined which bytes of the address were in the network portion and which were in the host portion. Today, an explicit mask identifies the network portion, and the boundary can fall between two adjacent bits, not just between bytes. However, the traditional classes are still used as defaults when no explicit division is specified.

Classes A, B, and C denote regular IP addresses. Classes D and E are used for multicasting and research addresses. Table 14.2 on the next page describes the characteristics of each class. The network portion of an address is denoted by N, and the host portion by H.

Table 14.2 Historical Internet address classes

Class	1st byte[a]	Format	Comments
A	1-127	N.H.H.H	Very early networks, or reserved for DoD
B	128-191	N.N.H.H	Large sites, usually subnetted, were hard to get
C	192-223	N.N.N.H	Easy to get, often obtained in sets
D	224-239	–	Multicast addresses, not permanently assigned
E	240-255	–	Experimental addresses

a. The value 0 is special and is not used as the first byte of regular IP addresses. 127 is reserved for the loopback address.

It's rare for a single physical network to have more than 100 computers attached to it, so class A and class B addresses (which allow for 16,777,214 hosts and 65,534 hosts per network, respectively) are really quite silly and wasteful. For example, the 127 class A networks use up half the available address space. Who knew that IPv4 address space would become so precious!

Subnetting

To make better use of these addresses, you can now reassign part of the host portion to the network portion by specifying an explicit 4-byte "subnet mask" or "netmask" in which the 1s correspond to the desired network portion and the 0s correspond to the host portion. The 1s must be leftmost and contiguous. At least eight bits must be allocated to the network part and at least two bits to the host part. Ergo, there are really only 22 possible values for an IPv4 netmask.

For example, the four bytes of a class B address would normally be interpreted as N.N.H.H. The implicit netmask for class B is therefore 255.255.0.0 in decimal notation. With a netmask of 255.255.255.0, however, the address would be interpreted as N.N.N.H. Use of the mask turns a single class B network address into 256 distinct class-C-like networks, each of which can support 254 hosts.

See page 478 for more information about ***ifconfig****.*

Netmasks are assigned with the **ifconfig** command as each network interface is set up. By default, **ifconfig** uses the inherent class of an address to figure out which bits are in the network part. When you set an explicit mask, you simply override this behavior.

Netmasks that do not end at a byte boundary can be annoying to decode and are often written as /XX, where XX is the number of bits in the network portion of the address. This is sometimes called CIDR (Classless Inter-Domain Routing; see page 460) notation. For example, the network address 128.138.243.0/26 refers to the first of four networks whose first bytes are 128.138.243. The other three networks have 64, 128, and 192 as their fourth bytes. The netmask associated with these networks is 255.255.255.192 or 0xFFFFFFC0; in binary, it's 26 ones followed by 6 zeros. Exhibit C breaks out these numbers in a bit more detail.

Exhibit C Netmask base conversion

IP address	128	.	138	.	243	.	0
Decimal netmask	255	.	255	.	255	.	192
Hex netmask	f f	.	f f	.	f f	.	c 0
Binary netmask	1111 1111	.	1111 1111	.	1111 1111	.	1100 0000

A /26 network has 6 bits left (32 – 26 = 6) to number hosts. 2^6 is 64, so the network has 64 potential host addresses. However, it can only accommodate 62 actual hosts because the all-0 and all-1 host addresses are reserved (they are the network and broadcast addresses, respectively).

In our 128.138.243.0/26 example, the extra two bits of network address obtained by subnetting can take on the values 00, 01, 10, and 11. The 128.138.243.0/24 network has thus been divided into four /26 networks:

- 128.138.243.0/26 (0 in decimal is **00**000000 in binary)
- 128.138.243.64/26 (64 in decimal is **01**000000 in binary)
- 128.138.243.128/26 (128 in decimal is **10**000000 in binary)
- 128.138.243.192/26 (192 in decimal is **11**000000 in binary)

The boldfaced bits of the last byte of each address are the bits that belong to the network portion of that byte.

Tricks and tools for subnet arithmetic

It's confusing to do all this bit twiddling in your head, but some tricks can make it simpler. The number of hosts per network and the value of the last byte in the netmask always add up to 256:

last netmask byte = 256 – net size

For example, 256 – 64 = 192, which is the final byte of the netmask in the preceding example. Another arithmetic fact is that the last byte of an actual network address (as opposed to a netmask) must be evenly divisible by the number of hosts per network. We see this fact in action in the 128.138.243.0/26 example, where the last bytes of the networks are 0, 64, 128, and 192—all divisible by 64.[7]

Given an IP address (say, 128.138.243.100), we cannot tell without the associated netmask what the network address and broadcast address will be. Table 14.3 on the next page shows the possibilities for /16 (the default for a class B address), /24 (a plausible value), and /26 (a reasonable value for a small network).

The network address and broadcast address steal two hosts from each network, so it would seem that the smallest meaningful network would have four possible

7. Of course, 0 counts as being divisible by any number…

Table 14.3 Example IPv4 address decodings

IP address	Netmask	Network	Broadcast
128.138.243.100/16	255.255.0.0	128.138.0.0	128.138.255.255
128.138.243.100/24	255.255.255.0	128.138.243.0	128.138.243.255
128.138.243.100/26	255.255.255.192	128.138.243.64	128.138.243.127

hosts: two real hosts—usually at either end of a point-to-point link—and the network and broadcast addresses. To have four values for hosts requires two bits in the host portion, so such a network would be a /30 network with netmask 255.255.255.252 or 0xFFFFFFFC. However, a /31 network is in fact treated as a special case (see RFC3021) and has no network or broadcast address; both of its two addresses are used for hosts, and its netmask is 255.255.255.254.

A handy web site called the IP Calculator by Krischan Jodies (it's available at jodies.de/ipcalc) helps with binary/hex/mask arithmetic. IP Calculator displays everything you might need to know about a network address and its netmask, broadcast address, hosts, etc. A tarball for a command-line version of the tool, **ipcalc**, is also available.

On Ubuntu you can install **ipcalc** through **apt-get**.

Here's some sample IP Calculator output, munged a bit to help with formatting:

```
Address:   24.8.175.69          00011000.00001000.10101111 .01000101
Netmask:   255.255.255.0 = 24   11111111.11111111.11111111 .00000000
Wildcard:  0.0.0.255            00000000.00000000.00000000 .11111111
=>
Network:   24.8.175.0/24        00011000.00001000.10101111 .00000000  (Class A)
Broadcast: 24.8.175.255         00011000.00001000.10101111 .11111111
HostMin:   24.8.175.1           00011000.00001000.10101111 .00000001
HostMax:   24.8.175.254         00011000.00001000.10101111 .11111110
```

The output provides both easy-to-understand versions of the addresses and "cut and paste" versions. Very useful.

Red Hat includes a similar but unrelated program that's also called **ipcalc**. However, it's relatively useless because it only understands default IP address classes.

If a dedicated IP calculator isn't available, the standard utility **bc** makes a good backup utility since it can do arithmetic in any base. Set the input and output bases with the **ibase** and **obase** directives. Set the **obase** first; otherwise, it's interpreted relative to the new **ibase**.

CIDR: Classless Inter-Domain Routing

CIDR is defined in RFC1519.

Like subnetting, of which it is a direct extension, CIDR relies on an explicit netmask to define the boundary between the network and host parts of an address. But unlike subnetting, CIDR allows the network portion to be made *smaller* than would be implied by an address's implicit class. A short CIDR mask may have the

effect of aggregating several networks for purposes of routing. Hence, CIDR is sometimes referred to as supernetting.

CIDR simplifies routing information and imposes hierarchy on the routing process. Although CIDR was only intended as an interim solution along the road to IPv6, it has proved to be sufficiently powerful to handle the Internet's growth problems for the better part of a decade.

For example, suppose that a site has been given a block of eight class C addresses numbered 192.144.0.0 through 192.144.7.0 (in CIDR notation, 192.144.0.0/21). Internally, the site could use them as

- 1 network of length /21 with 2,046 hosts, netmask 255.255.248.0
- 8 networks of length /24 with 254 hosts each, netmask 255.255.255.0
- 16 networks of length /25 with 126 hosts each, netmask 255.255.255.128
- 32 networks of length /26 with 62 hosts each, netmask 255.255.255.192

and so on. But from the perspective of the Internet, it's not necessary to have 32, 16, or even 8 routing table entries for these addresses. They all refer to the same organization, and all the packets go to the same ISP. A single routing entry for 192.144.0.0/21 suffices. CIDR makes it easy to allocate portions of class A and B addresses and thus increases the number of available addresses manyfold.

Inside your network, you can mix and match regions of different subnet lengths as long as all the pieces fit together without overlaps. This is called variable length subnetting. For example, an ISP with the 192.144.0.0/21 allocation could define some /30 networks for point-to-point customers, some /24s for large customers, and some /27s for smaller folks.

All the hosts on a network must be configured with the same netmask. You can't tell one host that it is a /24 and another host on the same network that it is a /25.

Address allocation

Only network numbers are formally assigned; sites must define their own host numbers to form complete IP addresses. You can subdivide the address space that has been assigned to you into subnets in whatever manner you like.

Administratively, ICANN (the Internet Corporation for Assigned Names and Numbers) has delegated blocks of addresses to five regional Internet registries, and these regional authorities are responsible for doling out subblocks to ISPs within their regions (see Table 14.4 on the next page). These ISPs in turn divide up their blocks and hand out pieces to individual clients. Only large ISPs should ever have to deal directly with one of the ICANN-sponsored address registries.

The delegation from ICANN to regional registries and then to national or regional ISPs has allowed for further aggregation in the backbone routing tables. ISP customers who have been allocated address space within the ISP's block do not need individual routing entries on the backbone. A single entry for the aggregated block that points to the ISP suffices.

Table 14.4 Regional Internet registries

Name	Site	Region covered
ARIN	arin.net	North America, part of the Caribbean
APNIC	apnic.net	Asia/Pacific region, including Australia and New Zealand
AfriNIC	afrinic.net	Africa
LACNIC	lacnic.net	Central and South America, part of the Caribbean
RIPE NCC	ripe.net	Europe and surrounding areas

Private addresses and network address translation (NAT)

Another factor that has helped decelerate the rate at which IPv4 addresses are consumed is the use of private IP address spaces, described in RFC1918. These addresses are used by your site internally but are never shown to the Internet (or at least, not intentionally). A border router translates between your private address space and the address space assigned by your ISP.

RFC1918 sets aside 1 class A network, 16 class B networks, and 256 class C networks that will never be globally allocated and can be used internally by any site. Table 14.5 shows the options. (The "CIDR range" column shows each range in the more compact CIDR notation; it does not add additional information.)

Table 14.5 IP addresses reserved for private use

IP class	From	To	CIDR range
Class A	10.0.0.0	10.255.255.255	10.0.0.0/8
Class B	172.16.0.0	172.31.255.255	172.16.0.0/12
Class C	192.168.0.0	192.168.255.255	192.168.0.0/16

The original idea was that sites would choose an address class from among these options to fit the size of their organizations. But now that CIDR and subnetting are universal, it probably makes the most sense to use the class A address (subnetted, of course) for all new private networks.

To allow hosts that use these private addresses to talk to the Internet, the site's border router runs a system called NAT (Network Address Translation). NAT intercepts packets addressed with these internal addresses and rewrites their source addresses, using a real external IP address and perhaps a different source port number. It also maintains a table of the mappings it has made between internal and external address/port pairs so that the translation can be performed in reverse when answering packets arrive from the Internet.

NAT's use of port number mapping multiplexes several conversations onto the same IP address so that a single external address can be shared by many internal hosts. In some cases, a site can get by with only one "real" IP address. For example,

this is the default configuration for most mass-market routers used with cable and DSL modems.

A site that uses NAT must still request a small section of address space from its ISP, but most of the addresses thus obtained are used for NAT mappings and are not assigned to individual hosts. If the site later wants to choose another ISP, only the border router and its NAT configuration need be updated, not the configurations of the individual hosts.

Large organizations that use NAT and RFC1918 addresses must institute some form of central coordination so that all hosts, independently of their department or administrative group, have unique IP addresses. The situation can become complicated when one company that uses RFC1918 address space acquires or merges with another company that's doing the same thing. Parts of the combined organization must often renumber.

It is possible to have a UNIX or Linux box perform the NAT function, but most sites prefer to delegate this task to their routers or network connection devices.[8] See the vendor-specific sections later in this chapter for details.

An incorrect NAT configuration can let private-address-space packets escape onto the Internet. The packets may get to their destinations, but answering packets won't be able to get back. CAIDA,[9] an organization that collects operational data from the Internet backbone, finds that 0.1% to 0.2% of the packets on the backbone have either private addresses or bad checksums. This sounds like a tiny percentage, but it represents thousands of packets every minute on a busy circuit. See caida.org for other interesting statistics and network measurement tools.

One issue raised by NAT is that an arbitrary host on the Internet cannot initiate connections to your site's internal machines. To get around this limitation, NAT implementations let you preconfigure externally visible "tunnels" that connect to specific internal hosts and ports.[10]

Another issue is that some applications embed IP addresses in the data portion of packets; these applications are foiled or confused by NAT. Examples include some media streaming systems, routing protocols, and FTP commands. NAT sometimes breaks VPNs (virtual private networks), too.

NAT hides interior structure. This secrecy feels like a security win, but the security folks say NAT doesn't really help for security and does not replace the need for a firewall. Unfortunately, NAT also foils attempts to measure the size and

8. Of course, many routers now run embedded Linux kernels. Even so, these dedicated systems are still generally more proficient and more secure than general-purpose computers that also forward packets.
9. CAIDA, pronounced "kay duh," is the Cooperative Association for Internet Data Analysis at the San Diego Supercomputer Center on the UCSD campus (caida.org).
10. Many routers also support the Universal Plug and Play (UPnP) standards promoted by Microsoft, one feature of which allows interior hosts to set up their own dynamic NAT tunnels. This can be either a godsend or a security risk, depending on your perspective. The feature is easily disabled at the router if you wish to do so.

topology of the Internet. See RFC4864, *Local Network Protection for IPv6*, for a good discussion of both the real and illusory benefits of NAT in IPv4.

IPv6 addressing

IPv6 addresses are 128 bits long. These long addresses were originally intended to solve the problem of IP address exhaustion. But now that they're here, they are being exploited to help with issues of routing, mobility, and locality of reference.

IPv4 addresses were not designed to be geographically clustered in the manner of phone numbers or zip codes, but clustering was added after the fact in the form of the CIDR conventions. (Of course, the relevant "geography" is really routing space rather than physical location.) CIDR was so technically successful that hierarchical subassignment of network addresses is now assumed throughout IPv6. Your IPv6 ISP assigns you an address prefix that you simply prepend to the local parts of your addresses, usually at your border router.

The boundary between the network portion and the host portion of an IPv6 address is fixed at /64, so there can be no disagreement or confusion about how long an address's network portion "really" is. Stated another way, true subnetting no longer exists in the IPv6 world, although the term "subnet" lives on as a synonym for "local network." Even though network numbers are always 64 bits long, routers needn't pay attention to all 64 bits when making routing decisions. They can route packets based on prefixes, just as they do under CIDR.

An early scheme outlined in RFC2374 called for four standardized subdivision levels within the network portion of an IPv6 address. But in light of the positive experience with letting ISPs manage their own IPv4 address subdivisions, that plan was withdrawn in RFC3587. ISPs are now free to set delegation boundaries wherever they wish.

The 64-bit host ID can potentially be derived from the hardware interface's 48-bit MAC address.[11] This scheme allows for automatic host numbering, which is a nice feature for sysadmins since only the subnet needs to be managed.

The fact that the MAC address can be seen at the IP layer has both good and bad implications. The good part is that host number configuration can be completely automatic. The bad part is that the brand and model of interface card are encoded in the first half of the MAC address, so prying eyes and hackers with code for a particular architecture will be helped along. The IPv6 standards point out that sites are not *required* to use MAC addresses to derive host IDs; they can use whatever numbering system they want.

11. More specifically, it is the MAC address with the two bytes 0xFFFE inserted in the middle and one bit (bit 6 of the first byte, numbering bits from the left, starting at 0) complemented; see RFC4291. The standard for converting 48-bit MAC addresses into 64-bit IP host numbers is known as EUI-64.

Here are some useful sources of additional IPv6 information:

- ipv6tf.org – An IPv6 information portal
- ipv6.org – FAQs and technical information
- ipv6forum.com – Marketing folks and IPv6 propaganda

- RFC3587 – *IPv6 Global Unicast Address Format*
- RFC4291 – *IP Version 6 Addressing Architecture*

Various schemes have been proposed to ease the transition from IPv4 to IPv6, mostly focusing on ways to tunnel IPv6 traffic through the IPv4 network to compensate for gaps in IPv6 support. The two tunneling systems in common use are called 6to4 and Teredo; the latter, named after a family of wood-boring shipworms, can be used on systems behind a NAT device.

14.5 ROUTING

Routing is the process of directing a packet through the maze of networks that stand between its source and its destination. In the TCP/IP system, it is similar to asking for directions in an unfamiliar country. The first person you talk to might point you toward the right city. Once you were a bit closer to your destination, the next person might be able to tell you how to get to the right street. Eventually, you get close enough that someone can identify the building you're looking for.

Routing information takes the form of rules ("routes"), such as "To reach network A, send packets through machine C." There can also be a default route that tells what to do with packets bound for a network to which there is no explicit route.

Routing information is stored in a table in the kernel. Each table entry has several parameters, including a mask for each listed network. To route a packet to a particular address, the kernel picks the most specific of the matching routes—that is, the one with the longest mask. If the kernel finds no relevant route and no default route, then it returns a "network unreachable" ICMP error to the sender.

The word "routing" is commonly used to mean two distinct things:

- Looking up a network address in the routing table to forward a packet toward its destination
- Building the routing table in the first place

In this section we examine the forwarding function and look at how routes can be manually added to or deleted from the routing table. We defer the more complicated topic of routing protocols that build and maintain the routing table until Chapter 15.

Routing tables

You can examine a machine's routing table with **netstat -r**. Use **netstat -rn** to avoid DNS lookups and present all the information numerically, which is generally more useful. We discuss **netstat** in more detail starting on page 868, but here is a short example to give you a better idea of what routes look like:

```
redhat$ netstat -rn
Kernel IP routing table
Destination     Genmask         Gateway         Fl  MSS  Iface
132.236.227.0   255.255.255.0   132.236.227.93  U   1500 eth0
default         0.0.0.0         132.236.227.1   UG  1500 eth0
132.236.212.0   255.255.255.192 132.236.212.1   U   1500 eth1
132.236.220.64  255.255.255.192 132.236.212.6   UG  1500 eth1
127.0.0.1       255.255.255.255 127.0.0.1       U   3584 lo
```

This host has two network interfaces: 132.236.227.93 (eth0) on the network 132.236.227.0/24 and 132.236.212.1 (eth1) on the network 132.236.212.0/26.

The destination field is usually a network address, although you can also add host-specific routes (their genmask is 255.255.255.255 since all bits are consulted). An entry's gateway field must contain the full IP address of a local network interface or adjacent host; on Linux kernels it can be 0.0.0.0 to invoke the default gateway.

For example, the fourth route in the table above says that to reach the network 132.236.220.64/26, packets must be sent to the gateway 132.236.212.6 through interface eth1. The second entry is a default route; packets not explicitly addressed to any of the three networks listed (or to the machine itself) are sent to the default gateway host, 132.236.227.1.

A host can only route packets to gateway machines that are reachable through a directly connected network. The local host's job is limited to moving packets one hop closer to their destinations, so it is pointless to include information about nonadjacent gateways in the local routing table. Each gateway that a packet visits makes a fresh next-hop routing decision based on its own local routing database.[12]

*See page 481 for more information about the **route** command.*

Routing tables can be configured statically, dynamically, or with a combination of the two approaches. A static route is one that you enter explicitly with the **route** command. Static routes remain in the routing table as long as the system is up; they are often set up at boot time from one of the system startup scripts. For example, the Linux commands

```
# route add -net 132.236.220.64 netmask 255.255.255.192
    gw 132.236.212.6 eth1
# route add default gw 132.236.227.1 eth0
```

12. The IP source routing feature is an exception to this rule; see page 473.

add the fourth and second routes displayed by **netstat -rn** above. (The first and third routes in that display were added by **ifconfig** when the eth0 and eth1 interfaces were configured.)

The final route is also added at boot time. It configures the loopback interface, which prevents packets sent from the host to itself from going out on the network. Instead, they are transferred directly from the network output queue to the network input queue inside the kernel.

In a stable local network, static routing is an efficient solution. It is easy to manage and reliable. However, it requires that the system administrator know the topology of the network accurately at boot time and that the topology not change often.

Most machines on a local area network have only one way to get out to the rest of the network, so the routing problem is easy. A default route added at boot time suffices to point toward the way out. Hosts that use DHCP (see page 469) to get their IP addresses can also obtain a default route with DHCP.

For more complicated network topologies, dynamic routing is required. Dynamic routing is implemented by a daemon process that maintains and modifies the routing table. Routing daemons on different hosts communicate to discover the topology of the network and to figure out how to reach distant destinations. Several routing daemons are available. See Chapter 15, *Routing*, for details.

ICMP redirects

Although IP generally does not concern itself with the management of routing information, it does define a naive damage control feature called an ICMP redirect. When a router forwards a packet to a machine on the same network from which the packet was originally received, something is clearly wrong. Since the sender, the router, and the next-hop router are all on the same network, the packet could have been forwarded in one hop rather than two. The router can conclude that the sender's routing tables are inaccurate or incomplete.

In this situation, the router can notify the sender of its problem by sending an ICMP redirect packet. In effect, a redirect says, "You should not be sending packets for host *xxx* to me; you should send them to host *yyy* instead."

In theory, the recipient of a redirect can adjust its routing table to fix the problem. In practice, redirects contain no authentication information and are therefore untrustworthy. Dedicated routers usually ignore redirects, but most UNIX and Linux systems accept them and act on them by default. You'll need to consider the possible sources of redirects in your network and disable their acceptance if they could pose a problem.

Under Linux, the variable **accept_redirects** in the **/proc** hierarchy controls the acceptance of ICMP redirects. See page 504 for instructions on examining and resetting this variable.

On Solaris, use **ndd -set /dev/ip ip_ignore_redirect 1** to disregard ICMP redirects. See page 498 for more details.

Although HP-UX also uses the **ndd** command to control its IP protocol stack, the underlying IP implementation lacks the ability to ignore ICMP redirects. However, you can arrange to have the routes that result from these redirects deleted from the routing table a second later with

```
ndd -set /dev/ip ip_ire_redirect_interval 1000
```

Some versions of HP-UX have enforced minima of 5 or 60 seconds on this parameter (which is expressed in milliseconds), but HP-UX 11 appears to accept smaller values without complaint.

On AIX, the command to ignore ICMP redirects is **no -p -o ipignoreredirects=1**. The **-p** option makes it a permanent change; omit this to test the change temporarily. See page 507 for more details.

14.6 ARP: THE ADDRESS RESOLUTION PROTOCOL

ARP is defined in RFC826.

Although IP addresses are hardware-independent, hardware addresses must still be used to actually transport data across a network's link layer.[13] ARP, the Address Resolution Protocol, discovers the hardware address associated with a particular IP address. It can be used on any kind of network that supports broadcasting but is most commonly described in terms of Ethernet.

If host A wants to send a packet to host B on the same Ethernet, it uses ARP to discover B's hardware address. If B is not on the same network as A, host A uses the routing system to determine the next-hop router along the route to B and then uses ARP to find that router's hardware address. Since ARP uses broadcast packets, which cannot cross networks,[14] it can only be used to find the hardware addresses of machines directly connected to the sending host's local network.

Every machine maintains a table in memory called the ARP cache, which contains the results of recent ARP queries. Under normal circumstances, many of the addresses a host needs are discovered soon after booting, so ARP does not account for a lot of network traffic.

ARP works by broadcasting a packet of the form "Does anyone know the hardware address for 128.138.116.4?" The machine being searched for recognizes its own IP address and replies, "Yes, that's the IP address assigned to one of my network interfaces, and the corresponding Ethernet address is 8:0:20:0:fb:6a."

The original query includes the IP and Ethernet addresses of the requestor so that the machine being sought can reply without issuing an ARP query of its own.

13. Except on point-to-point links, on which the identity of the destination is sometimes implicit.
14. Routers can in fact be configured to flood broadcast packets to other networks, but this is generally a bad idea. If you find yourself wanting to forward broadcasts, there is most likely something amiss with your network or server architecture.

Thus, the two machines learn each other's ARP mappings with only one exchange of packets. Other machines that overhear the requestor's initial broadcast can record its address mapping, too.

The **arp** command examines and manipulates the kernel's ARP cache, adds or deletes entries, and flushes or shows the table. **arp -a** displays the contents of the ARP cache; output formats vary.

The **arp** command is generally useful only for debugging and for situations that involve special hardware. For example, if two hosts on a network are using the same IP address, one has the right ARP table entry and one is wrong. You can use the **arp** command to track down the offending machine.

14.7 DHCP: THE DYNAMIC HOST CONFIGURATION PROTOCOL

DHCP is defined in RFCs 2131 and 2132.

When you plug a device or computer into a network, it usually obtains an IP address for itself on the local network, sets up an appropriate default route, and connects itself to a local DNS server. The Dynamic Host Configuration Protocol (DHCP) is the hidden Svengali that makes this magic happen.

The protocol lets a DHCP client "lease" a variety of network and administrative parameters from a central server that is authorized to distribute them. The leasing paradigm is particularly convenient for PCs that are turned off when not in use and for networks that must support transient guests such as laptops.

Leasable parameters include

- IP addresses and netmasks
- Gateways (default routes)
- DNS name servers
- Syslog hosts
- WINS servers, X font servers, proxy servers, NTP servers
- TFTP servers (for loading a boot image)

There are dozens more—see RFC2132. Real-world use of the more exotic parameters is rare, however.

Clients must report back to the DHCP server periodically to renew their leases. If a lease is not renewed, it eventually expires. The DHCP server is then free to assign the address (or whatever was being leased) to a different client. The lease period is configurable, but it's usually quite long (hours or days).

Even if you want each host to have its own permanent IP address, DHCP can save you time and suffering. Once the server is up and running, clients can use it to obtain their network configuration at boot time. No fuss, no mess, and most importantly, a minimum of local configuration on the client machines.

DHCP software

ISC, the Internet Systems Consortium, maintains a very nice open source reference implementation of DHCP. Major versions 2, 3, and 4 of ISC's software are all in common use, and all of these versions work fine for basic service. Version 3 supports backup DHCP servers, and version 4 supports IPv6. Server, client, and relay agents are all available from isc.org.

Major Linux distributions all use some version of the ISC software, although you may have to install the server portion explicitly. The server package is called **dhcp** on Red Hat, **dhcp3-server** on Ubuntu, and **dhcp-server** on SUSE.

Non-Linux systems often have their own home-grown DCHP implementations, and unfortunately all our example UNIX systems fall into this category.

It's best not to tamper with the client side of DHCP, since that part of the code is relatively simple and comes preconfigured and ready to use. Changing the client side of DHCP is not trivial.

However, if you need to run a DHCP *server*, we recommend the ISC package over vendor-specific implementations. In a typical heterogeneous network environment, administration is greatly simplified by standardizing on a single implementation. The ISC software provides a reliable, open source solution that builds without incident on most versions of UNIX.

In the next few sections, we briefly discuss the DHCP protocol, explain how to set up the ISC server that implements it, and review some client configuration issues.

How DHCP works

DHCP is a backward-compatible extension of BOOTP, a protocol originally devised to help diskless UNIX workstations boot. DHCP generalizes the parameters that can be supplied and adds the concept of a lease period for assigned values.

A DHCP client begins its interaction with a DHCP server by broadcasting a "Help! Who am I?" message.[15] If a DHCP server is present on the local network, it negotiates with the client to provide an IP address and other networking parameters. If there is no DHCP server on the local net, servers on different subnets can receive the initial broadcast message through a separate piece of DHCP software that acts as a relay agent.

When the client's lease time is half over, it attempts to renew its lease. The server is obliged to keep track of the addresses it has handed out, and this information must persist across reboots. Clients are supposed to keep their lease state across reboots too, although many do not. The goal is to maximize stability in network configuration. In theory, all software should be prepared for network configurations to change at a moment's notice, but a lot of software still makes unwarranted assumptions about the continuity of the network.

15. Clients initiate conversations with the DHCP server by using the generic all-ones broadcast address. The clients don't yet know their subnet masks and therefore can't use the subnet broadcast address.

ISC's DHCP software

ISC's server daemon is called **dhcpd**, and its configuration file is **dhcpd.conf**, usually found in **/etc** or **/etc/dhcp3**. The format of the config file is a bit fragile; leave out a semicolon and you may receive a cryptic, unhelpful error message.

When setting up a new DHCP server, you must also make sure that an empty lease database file has been created. Check the summary at the end of the man page for **dhcpd** to find the correct location for the lease file on your system. It's usually somewhere underneath **/var**.

To set up the **dhcpd.conf** file, you need the following information:

- The subnets for which **dhcpd** should manage IP addresses, and the ranges of addresses to dole out
- A list of static IP address assignments you want to make (if any), along with the MAC (hardware) addresses of the recipients
- The initial and maximum lease durations, in seconds
- Any other options the server should pass to DHCP clients: netmask, default route, DNS domain, name servers, etc.

The **dhcpd** man page outlines the configuration process, and the **dhcpd.conf** man page covers the exact syntax of the config file. In addition to setting up your configuration, make sure **dhcpd** is started automatically at boot time. (See Chapter 3, *Booting and Shutting Down*, for instructions.) It's helpful to make startup of the daemon conditional on the existence of the **dhcpd.conf** file if your system doesn't do this for you automatically.

Below is a sample **dhcpd.conf** file from a Linux box with two interfaces, one internal and one that connects to the Internet. This machine performs NAT translation for the internal network (see page 462) and leases out a range of 10 IP addresses on this network as well.

Every subnet must be declared, even if no DHCP service is provided on it, so this **dhcpd.conf** file contains a dummy entry for the external interface. It also includes a `host` entry for one particular machine that needs a fixed address.

```
# global options

option domain-name "synack.net";
option domain-name-servers gw.synack.net;
option subnet-mask 255.255.255.0;
default-lease-time 600;
max-lease-time 7200;

subnet 192.168.1.0 netmask 255.255.255.0 {
      range 192.168.1.51 192.168.1.60;
      option broadcast-address 192.168.1.255;
      option routers gw.synack.net;
}
```

```
subnet 209.180.251.0 netmask 255.255.255.0 {
}

host gandalf {
    hardware ethernet 08:00:07:12:34:56;
    fixed-address gandalf.synack.net;
}
```

See Chapter 17 for more information about DNS.

Unless you make static IP address assignments such as the one for gandalf above, you'll need to consider how your DHCP configuration will interact with DNS. The easy option is to assign a generic name to each dynamically leased address (e.g., dhcp1.synack.net) and allow the names of individual machines to float along with their IP addresses. Alternatively, you can configure **dhcpd** to update the DNS database as it hands out addresses. The dynamic update solution is more complicated, but it has the advantage of preserving each machine's hostname.

ISC's DHCP relay agent is a separate daemon called **dhcrelay**. It's a simple program with no configuration file of its own, although Linux distributions often add a startup harness that feeds it the appropriate command-line arguments for your site. **dhcrelay** listens for DHCP requests on local networks and forwards them to a set of remote DHCP servers that you specify. It's handy both for centralizing the management of DHCP service and for provisioning backup DHCP servers.

ISC's DHCP client is similarly configuration free. It stores status files for each connection in the directory **/var/lib/dhcp** or **/var/lib/dhclient**. The files are named after the interfaces they describe. For example, **dhclient-eth0.leases** would contain all the networking parameters that **dhclient** had set up on behalf of the eth0 interface.

14.8 SECURITY ISSUES

We address the topic of security in a chapter of its own (Chapter 22), but several security issues relevant to IP networking merit discussion here. In this section, we briefly look at a few networking features that have acquired a reputation for causing security problems and recommend ways to minimize their impact. The details of our example systems' default behavior on these issues (and the appropriate methods for changing them) vary considerably and are discussed in the system-specific material starting on page 484.

IP forwarding

A UNIX or Linux system that has IP forwarding enabled can act as a router. That is, it can accept third-party packets on one network interface, match them to a gateway or destination host on another interface, and retransmit the packets.

Unless your system has multiple network interfaces and is actually supposed to function as a router, it's advisable to turn this feature off. Hosts that forward packets can sometimes be coerced into compromising security by making external

packets appear to have come from inside your network. This subterfuge can help an intruder's packets evade network scanners and packet filters.

It is perfectly acceptable for a host to use multiple network interfaces for its own traffic without forwarding third-party traffic.

ICMP redirects

ICMP redirects (see page 467) can maliciously reroute traffic and tamper with your routing tables. Most operating systems listen to ICMP redirects and follow their instructions by default. It would be bad if all your traffic were rerouted to a competitor's network for a few hours, especially while backups were running! We recommend that you configure your routers (and hosts acting as routers) to ignore and perhaps log ICMP redirect attempts.

Source routing

IP's source routing mechanism lets you specify an explicit series of gateways for a packet to transit on the way to its destination. Source routing bypasses the next-hop routing algorithm that's normally run at each gateway to determine how a packet should be forwarded.

Source routing was part of the original IP specification; it was intended primarily to facilitate testing. It can create security problems because packets are often filtered according to their origin. If someone can cleverly route a packet to make it appear to have originated within your network instead of the Internet, it might slip through your firewall. We recommend that you neither accept nor forward source-routed packets.

Broadcast pings and other directed broadcasts

Ping packets addressed to a network's broadcast address (instead of to a particular host address) are typically delivered to every host on the network. Such packets have been used in denial of service attacks; for example, the so-called Smurf attacks. (The "Smurf attacks" Wikipedia article has details.)

Broadcast pings are a form of "directed broadcast" in that they are packets sent to the broadcast address of a distant network. The default handling of such packets has been gradually changing. For example, versions of Cisco's IOS up through 11.x forwarded directed broadcast packets by default, but IOS releases since 12.0 do not. It is usually possible to convince your TCP/IP stack to ignore broadcast packets that come from afar, but since this behavior must be set on each interface, the task can be nontrivial at a large site.

IP spoofing

The source address on an IP packet is normally filled in by the kernel's TCP/IP implementation and is the IP address of the host from which the packet was sent. However, if the software creating the packet uses a raw socket, it can fill in any source address it likes. This is called IP spoofing and is usually associated with

some kind of malicious network behavior. The machine identified by the spoofed source IP address (if it is a real address at all) is often the victim in the scheme. Error and return packets can disrupt or flood the victim's network connections.

You should deny IP spoofing at your border router by blocking outgoing packets whose source address is not within your address space. This precaution is especially important if your site is a university where students like to experiment and may be tempted to carry out digital vendettas.

If you are using private address space internally, you can filter at the same time to catch any internal addresses escaping to the Internet. Such packets can never be answered (because they lack a backbone route) and always indicate that your site has an internal configuration error.

In addition to detecting outbound packets with bogus source addresses, you must also protect against a attacker's forging the source address on external packets to fool your firewall into thinking that they originated on your internal network. A heuristic known as "unicast reverse path forwarding" (uRPF) helps with this. It makes IP gateways discard packets that arrive on an interface that is different from the one on which they would be transmitted if the source address were the destination. It's a quick sanity check that uses the normal IP routing table as a way to validate the origin of network packets. Dedicated routers implement uRPF, but so does the Linux kernel. On Linux, it's enabled by default.

If your site has multiple connections to the Internet, it may be perfectly reasonable for inbound and outbound routes to be different. In this situation, you'll have to turn off uRPF to make your routing work properly. If your site has only one way out to the Internet, then turning on uRPF is usually safe and appropriate.

Host-based firewalls

Traditionally, a network packet filter or firewall connects your local network to the outside world and controls traffic according to a site-wide policy. Unfortunately, Microsoft has warped everyone's perception of how a firewall should work with its notoriously insecure Windows systems. The last few Windows releases all come with their own personal firewalls, and they complain bitterly if you try to turn the firewall off.

Our example systems all include packet filtering software, but you should not infer from this that every UNIX or Linux machine needs its own firewall. It does not. The packet filtering features are there to allow these machines to serve as network gateways.

However, we don't recommend using a workstation as a firewall. Even with meticulous hardening, full-fledged operating systems are too complex to be fully trustworthy. Dedicated network equipment is more predictable and more reliable—even if it secretly runs Linux.

Even sophisticated software solutions like those offered by Check Point (whose products run on UNIX, Linux, and Windows hosts) are not as secure as a dedicated device such as Cisco's Adaptive Security Appliance series. The software-only solutions are nearly the same price, to boot.

A more thorough discussion of firewall-related issues begins on page 932.

Virtual private networks

Many organizations that have offices in several locations would like to have all those locations connected to one big private network. Such organizations can use the Internet as if it were a private network by establishing a series of secure, encrypted "tunnels" among their various locations. A network that includes such tunnels is known as a virtual private network or VPN.

VPN facilities are also needed when employees must connect to your private network from their homes or from the field. A VPN system doesn't eliminate every possible security issue relating to such ad hoc connections, but it's secure enough for many purposes.

See page 943 for more information about IPsec.

Some VPN systems use the IPsec protocol, which was standardized by the IETF in 1998 as a relatively low-level adjunct to IP. Others, such as OpenVPN, implement VPN security on top of TCP using Transport Layer Security (TLS), formerly known as the Secure Sockets Layer (SSL). TLS is also on the IETF's standards track, although it hasn't yet been fully adopted.

A variety of proprietary VPN implementations are also available. These systems generally don't interoperate with each other or with the standards-based VPN systems, but that's not necessarily a major drawback if all the endpoints are under your control.

The TLS-based VPN solutions seem to be the marketplace winners at this point. They are just as secure as IPsec and considerably less complicated. Having a free implementation in the form of OpenVPN doesn't hurt either. (Unfortunately, it doesn't run on HP-UX or AIX yet.)

To support home and portable users, a common paradigm is for users to download a small Java or ActiveX component through their web browser. This component then provides VPN connectivity back to the enterprise network. The mechanism is convenient for users, but be aware that the browser-based systems differ widely in their implementations: some provide VPN service through a pseudo-network-interface, while others forward only specific ports. Still others are little more than glorified web proxies.

Be sure you understand the underlying technology of the solutions you're considering, and don't expect the impossible. True VPN service (that is, full IP-layer connectivity through a network interface) requires administrative privileges and software installation on the client, whether that client is Windows or a UNIX

laptop. Check browser compatibility too, since the voodoo involved in implementing browser-based VPN solutions often doesn't translate among browsers.

14.9 PPP: THE POINT-TO-POINT PROTOCOL

PPP is defined in RFC1331.

PPP represents an underlying communication channel as a virtual network interface. However, since the underlying channel need not have any of the features of an actual network, communication is restricted to the two hosts at the ends of the link—a virtual network of two. PPP has the distinction of being used on both the slowest and the fastest IP links, but for different reasons.

In its asynchronous form, PPP is best known as the protocol used to provide dial-up Internet service over phone lines and serial links. These channels are not inherently packet oriented, so the PPP device driver encodes network packets into a unified data stream and adds link-level headers and markers to separate packets.

In its synchronous form, PPP is the encapsulation protocol used on high-speed circuits that have routers at either end. It's also commonly used as part of the implementation of DSL and cable modems for broadband service. In these latter situations, PPP not only converts the underlying network system (often ATM in the case of DSL) to an IP-friendly form, but it also provides authentication and access control for the link itself. In a surreal, down-the-rabbit-hole twist, PPP can implement Ethernet-like semantics on top of an actual Ethernet, a configuration known as "PPP over Ethernet" or PPPoE.

Designed by committee, PPP is the "everything *and* the kitchen sink" encapsulation protocol. In addition to specifying how the link is established, maintained, and torn down, PPP implements error checking, authentication, encryption, and compression. These features make it adaptable to a variety of situations.

PPP as a dial-up technology was once an important topic for UNIX and Linux system administrators, but the widespread availability of broadband has made dial-up configuration largely irrelevant. At the same time, the high-end applications of PPP have mostly retreated into various pieces of dedicated network hardware. These days, the primary use of PPP is to connect through cellular modems.

14.10 BASIC NETWORK CONFIGURATION

Only a few steps are involved in adding a new machine to an existing local area network, but every system does it slightly differently. Systems typically provide a control panel GUI for basic network configuration, but more elaborate (or automated) setups may require you to edit the configuration files directly.

Before bringing up a new machine on a network that is connected to the Internet, secure it (Chapter 22, *Security*) so that you are not inadvertently inviting attackers onto your local network.

The basic steps to add a new machine to a local network are as follows:

- Assign a unique IP address and hostname.
- Make sure network interfaces are properly configured at boot time.
- Set up a default route and perhaps fancier routing.
- Point to a DNS name server to allow access to the rest of the Internet.

If you rely on DHCP for basic provisioning, most of the configuration chores for a new machine are performed on the DHCP server rather than on the new machine itself. New OS installations typically default to getting their configuration through DHCP, so new machines may require no network configuration at all. Refer to the DHCP section starting on page 469 for general information.

After any change that might affect booting, you should always reboot to verify that the machine comes up correctly. Six months later when the power has failed and the machine refuses to boot, it's hard to remember what change you made that might have caused the problem. (Refer also to Chapter 21, *Network Management and Debugging*.)

The process of designing and installing a physical network is touched on in Chapter 16, *Network Hardware*. If you are dealing with an existing network and have a general idea of how it is set up, it may not be necessary for you to read too much more about the physical aspects of networking unless you plan to extend the existing network.

In this section, we review the various commands and issues involved in manual network configuration. This material is general enough to apply to any UNIX or Linux system. In the vendor-specific sections starting on page 484, we address the unique twists that distinguish UNIX from Linux and separate the various vendors' systems.

As you work through basic network configuration on any machine, you'll find it helpful to test your connectivity with basic tools such as **ping** and **traceroute**. Those tools are actually described in the *Network Management and Debugging* chapter; see the sections starting on page 861 for more details.

Hostname and IP address assignment

See Chapter 17 for more information about DNS.

Administrators have various heartfelt theories about how the mapping from hostnames to IP addresses is best maintained: through the **hosts** file, LDAP, the DNS system, or perhaps some combination of those options. The conflicting goals are scalability, consistency, and maintainability versus a system that is flexible enough to allow machines to boot and function when not all services are available. *Prioritizing sources of administrative information* starting on page 739 describes how the various options can be combined.

Another consideration you might take into account when designing your addressing system is the possible need to renumber your hosts in the future. Unless you are using RFC1918 private addresses (see page 462), your site's IP addresses may

change when you switch ISPs. Such a transition becomes daunting if you must visit each host on the network to reconfigure its address. To expedite renumbering, you can use hostnames in configuration files and confine address mappings to a few centralized locations such as the DNS database and your DHCP configuration files.

The **/etc/hosts** file is the oldest and simplest way to map names to IP addresses. Each line starts with an IP address and continues with the various symbolic names by which that address is known.

Here is a typical **/etc/hosts** file for the host lollipop:

```
127.0.0.1       localhost
192.108.21.48   lollipop.atrust.com lollipop loghost
192.108.21.254  chimchim-gw.atrust.com chimchim-gw
192.108.21.1    ns.atrust.com ns
192.225.33.5    licenses.atrust.com license-server
```

A minimalist version would contain only the first two lines. localhost is commonly the first entry in the **/etc/hosts** file; this entry is unnecessary on many systems, but it doesn't hurt to include it. IPv6 addresses can go in this file as well.

Because **/etc/hosts** contains only local mappings and must be maintained on each client system, it's best reserved for mappings that are needed at boot time (e.g., the host itself, the default gateway, and name servers). Use DNS or LDAP to find mappings for the rest of the local network and the rest of the world. You can also use **/etc/hosts** to specify mappings that you do not want the rest of the world to know about and therefore do not publish in DNS.[16]

The **hostname** command assigns a hostname to a machine. **hostname** is typically run at boot time from one of the startup scripts, which obtains the name to be assigned from a configuration file. (Of course, each system does this slightly differently. See the system-specific sections beginning on page 484 for details.) The hostname should be fully qualified: that is, it should include both the hostname and the DNS domain name, such as anchor.cs.colorado.edu.

See page 728 for more information about LDAP.

At a small site, you can easily dole out hostnames and IP addresses by hand. But when many networks and many different administrative groups are involved, it helps to have some central coordination to ensure uniqueness. For dynamically assigned networking parameters, DHCP takes care of the uniqueness issues. Some sites now use LDAP databases to manage their hostnames and IP addresses assignments.

ifconfig: configure network interfaces

ifconfig enables or disables a network interface, sets its IP address and subnet mask, and sets various other options and parameters. It is usually run at boot time with command-line parameters taken from config files, but you can also run it by

16. You can also use a split DNS configuration to achieve this goal; see page 617.

hand to make changes on the fly. Be careful if you are making **ifconfig** changes and are logged in remotely—many a sysadmin has been locked out this way and had to drive in to fix things.

An **ifconfig** command most commonly has the form

ifconfig *interface* [*family*] *address options…*

For example, the command

```
ifconfig eth0 192.168.1.13 netmask 255.255.255.0 up
```

sets the IPv4 address and netmask associated with the interface eth0 and readies the interface for use.

interface identifies the hardware interface to which the command applies. It is usually a two- or three-character name followed by a number, but Solaris interface names can be longer. Some common names are ie0, le0, le1, ln0, en0, we0, qe0, hme0, eth0, and lan0. The loopback interface is lo on Linux and lo0 on Solaris, HP-UX, and AIX. On most systems, **ifconfig -a** lists the system's network interfaces and summarizes their current settings. Use **netstat -i** for this on HP-UX.

Under Solaris, network interfaces must be "attached" with **ifconfig** *interface* **plumb** before they become configurable and visible to **ifconfig -a**. You can use the **dladm** command to list interfaces regardless of whether they have been plumbed.

The *family* parameter tells **ifconfig** which network protocol ("address family") you want to configure. You can set up multiple protocols on an interface and use them all simultaneously, but they must be configured separately. The main options here are **inet** for IPv4 and **inet6** for IPv6; **inet** is assumed if you leave the parameter out. Linux systems support a handful of other legacy protocols such as AppleTalk and Novell IPX.

The *address* parameter specifies the interface's IP address. A hostname is also acceptable here, but the hostname must be resolvable to an IP address at boot time. For a machine's primary interface, this means that the hostname must appear in the local **hosts** file, since other name resolution methods depend on the network having been initialized.

The keyword **up** turns the interface on; **down** turns it off. When an **ifconfig** command assigns an IP address to an interface, as in the example above, the **up** parameter is implicit and does not need to be mentioned by name.

ifconfig understands lots of other options. The most common ones are mentioned below, but as always, consult your man pages for the final word on your particular system. **ifconfig** options all have symbolic names. Some options require an argument, which should be placed immediately after the option name and separated from the option name by a space.

The **netmask** option sets the subnet mask for the interface and is required if the network is not subnetted according to its address class (A, B, or C). The mask can

be specified in dotted decimal notation or as a 4-byte hexadecimal number beginning with **0x**. As usual, bits set to 1 are part of the network number, and bits set to 0 are part of the host number.

The **broadcast** option specifies the IP broadcast address for the interface, expressed in either hex or dotted quad notation. The default broadcast address is one in which the host part is set to all 1s. In the **ifconfig** example above, the autoconfigured broadcast address is 192.168.1.255.

You can set the broadcast address to any IP address that's valid for the network to which the host is attached. Some sites have chosen weird values for the broadcast address in the hope of avoiding certain types of denial of service attacks that are based on broadcast pings, but this is risky and probably overkill. Failure to properly configure every machine's broadcast address can lead to broadcast storms, in which packets travel from machine to machine until their TTLs expire.[17]

A better way to avoid problems with broadcast pings is to prevent your border routers from forwarding them and to tell individual hosts not to respond to them. See Chapter 22, *Security*, for instructions on how to implement these constraints.

Solaris integrates the **ifconfig** command with its DHCP client daemon. **ifconfig** *interface* **dhcp** configures the named interface with parameters leased from a local DHCP server, then starts **dhcpagent** to manage the leases over the long term. Other systems keep **ifconfig** ignorant of DHCP, with the DHCP software operating as a separate layer.

You can also get the configuration for a single interface with **ifconfig** *interface*:

```
solaris$ ifconfig e1000g0

e1000g0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500
     index 2 inet 192.168.10.10 netmask ffffff00 broadcast 192.168.10.255

redhat$ ifconfig eth0
eth0  Link encap:Ethernet  HWaddr 00:02:B3:19:C8:86
      inet addr:192.168.1.13  Bcast:192.168.1.255  Mask:255.255.255.0
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:206983 errors:0 dropped:0 overruns:0 frame:0
      TX packets:218292 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:100
      Interrupt:7 Base address:0xef00
```

The lack of collisions on the Ethernet interface in the second example may indicate a very lightly loaded network or, more likely, a switched network. On a shared

17. Broadcast storms occur because the same link-layer broadcast address must be used to transport packets no matter what the IP broadcast address has been set to. For example, suppose that machine X thinks the broadcast address is A1 and machine Y thinks it is A2. If X sends a packet to address A1, Y will receive the packet (because the link-layer destination address is the broadcast address), will see that the packet is not for itself and also not for the broadcast address (because Y thinks the broadcast address is A2), and may then forward the packet back to the net. If two machines are in Y's state, the packet circulates until its TTL expires. Broadcast storms can erode your bandwidth, especially on a large switched net.

network (one built with hubs instead of switches, or one that uses old-style coaxial Ethernet), check this number to ensure that it is below about 5% of the output packets. Lots of collisions indicate a loaded network that needs to be watched and possibly split into multiple subnets or migrated to a switched infrastructure.

Now that you know how to configure a network interface by hand, you need to figure out how the parameters to **ifconfig** are set when the machine boots, and you need to make sure that the new values are entered correctly. You normally do this by editing one or more configuration files; see the vendor-specific sections starting on page 484 for more information.

One additional comment regarding **ifconfig**: you can assign more than one IP address to an interface by making use of the concept of "virtual network interfaces" or "IP aliases." Administrators can do this to allow one machine to host several web sites. See page 967 for more information.

Network hardware options

Network hardware often has configurable options that are specific to its media type and have little to do with TCP/IP per se. One common example of this is modern-day Ethernet, wherein an interface card may support 10, 100, 1000, or even 10000 Mb/s in either half-duplex or full-duplex mode. Most equipment defaults to autonegotiation mode, in which both the card and its upstream connection (usually a switch port) try to guess what the other wants to use.

Historically, autonegotiation has worked about as well as a blindfolded cowpoke trying to rope a calf. Modern network devices play better together, but autonegotiation is still a common source of failure. High packet loss rates (especially for large packets) are a common artifact of failed autonegotiation.

If you're having problems with mysterious packet loss, turn off autonegotiation everywhere as your first course of action. Lock the interface speed and duplex on both servers and the switch ports to which they are connected. Autonegotiation is useful for ports in public areas where roving laptops may stop for a visit, but it serves no useful purpose for statically attached hosts other than avoiding a small amount of administration.

The exact method by which hardware options like autonegotiation are set varies widely, so we defer discussion of those details to the system-specific sections starting on page 484.

route: configure static routes

The **route** command defines static routes, explicit routing table entries that never change, even if you run a routing daemon. When you add a new machine to a local area network, you usually need to specify only a default route.

This book's discussion of routing is split between this section and Chapter 15, *Routing*. Although most of the basic information about routing and the **route**

command is found in this section, you might find it helpful to read the first few sections of Chapter 15 if you need more information.

Routing is performed at the IP layer. When a packet bound for some other host arrives, the packet's destination IP address is compared with the routes in the kernel's routing table. If the address matches a route in the table, the packet is forwarded to the next-hop gateway IP address associated with that route.

There are two special cases. First, a packet may be destined for some host on a directly connected network. In this case, the "next-hop gateway" address in the routing table is one of the local host's own interfaces, and the packet is sent directly to its destination. This type of route is added to the routing table for you by the **ifconfig** command when you configure an interface.

Second, it may be that no route matches the destination address. In this case, the default route is invoked if one exists. Otherwise, an ICMP "network unreachable" or "host unreachable" message is returned to the sender.

Many local area networks have only one way out, so all they need is a single default route that points to the exit. On the Internet backbone, the routers do not have default routes. If there is no routing entry for a destination, that destination cannot be reached.

Each **route** command adds or removes one route. Unfortunately, **route** is one of a handful of UNIX commands that function identically across systems and yet have somewhat different syntax everywhere. Here's a prototypical **route** command that works almost everywhere:

```
# route add -net 192.168.45.128/25 zulu-gw.atrust.net
```

This command adds a route to the 192.168.45.128/25 network through the gateway router zulu-gw.atrust.net, which must be either an adjacent host or one of the local host's own interfaces. (Linux requires the option name **gw** in front of the gateway address.) Naturally, **route** must be able to resolve zulu-gw.atrust.net into an IP address. Use a numeric IP address if your DNS server is on the other side of the gateway!

Linux also accepts an interface name (e.g., eth0) as the destination for a route. It has the same effect as specifying the interface's primary IP address as the gateway address. That is, the IP stack attempts direct delivery on that interface rather than forwarding to a separate gateway. Routing entries that were set up this way show their gateway addresses as 0.0.0.0 in **netstat -r** output. You can tell where the route really goes by looking in the `Iface` column for the interface name.

Destination networks were traditionally specified with separate IP addresses and netmasks, but all versions of **route** except that of HP-UX now understand CIDR notation (e.g., 128.138.176.0/20). CIDR notation is clearer and relieves you of the need to fuss over some of the system-specific syntax issues. Even Linux accepts CIDR notation, although the Linux man page for **route** doesn't admit this.

Solaris has a nifty **-p** option to **route** that makes your changes persistent across reboots. In addition to being entered in the kernel's routing table, the changes are recorded in **/etc/inet/static_routes** and restored at boot time.

Some other tricks:

- To inspect existing routes, use the command **netstat -nr**, or **netstat -r** if you want to see names instead of numbers. Numbers are often better if you are debugging, since the name lookup may be the thing that is broken. An example of **netstat** output is shown on page 466.
- Use the keyword **default** instead of an address or network name to set the system's default route.
- Use **route delete** or **route del** to remove entries from the routing table.
- UNIX systems use **route -f** or **route flush** to initialize the routing table and start over. Linux does not support this option.
- IPv6 routes are set up similarly to IPv4 routes. You'll need to tell **route** that you're working in IPv6 space with the **-inet6** or **-A inet6** option.
- **/etc/networks** maps names to network numbers, much like the **hosts** file maps hostnames to IP addresses. Commands such as **route** that expect a network number can accept a name if it is listed in the **networks** file. Network names can also be listed in an NIS database or in DNS; see RFC1101.
- You can use **route add -host** to set up a route that's specific to a single IP address. It's essentially the same as a route with a netmask of 255.255.255.255, but it's flagged separately in the routing table.

DNS configuration

To configure a machine as a DNS client, you need only set up the **/etc/resolv.conf** file. DNS service is not, strictly speaking, required (see page 739), but it's hard to imagine a situation in which you'd want to eliminate it completely.

The **resolv.conf** file lists the DNS domains that should be searched to resolve names that are incomplete (that is, not fully qualified, such as anchor instead of anchor.cs.colorado.edu) and the IP addresses of the name servers to contact for name lookups. A sample is shown here; for more details, see page 561.

```
search cs.colorado.edu colorado.edu
nameserver 128.138.242.1
nameserver 128.138.243.151
nameserver 192.108.21.1
```

/etc/resolv.conf should list the "closest" stable name server first. Servers are contacted in order, and the timeout after which the next server in line is tried can be quite long. You can have up to three `nameserver` entries. If possible, you should always have more than one.

If the local host obtains the addresses of its DNS servers through DHCP, the DHCP client software stuffs these addresses into the **resolv.conf** file for you when it obtains the leases. Since DHCP configuration is the default for most systems, you generally do not need to configure the **resolv.conf** file manually if your DHCP server has been set up correctly.

Many sites use Microsoft's Active Directory DNS server implementation. That works fine with the standard UNIX and Linux **resolv.conf**; there's no need to do anything differently.

14.11 SYSTEM-SPECIFIC NETWORK CONFIGURATION

On early UNIX systems, you configured the network by editing the system startup scripts and directly changing the commands they contained. Modern systems have read-only scripts; they cover a variety of configuration scenarios and choose among them by reusing information from other system files or consulting configuration files of their own.

Although this separation of configuration and implementation is a good idea, every system does it a little bit differently. The format and use of the **/etc/hosts** and **/etc/resolv.conf** files are relatively consistent among UNIX and Linux systems, but that's about all you can count on for sure.

Most systems provide some sort of GUI interface for basic configuration tasks, but the mapping between the visual interface and the configuration files behind the scenes is often unclear. In addition, the GUIs tend to ignore advanced configurations, and they are relatively inconvenient for remote and automated administration. In the next sections, we pick apart some of the variations among our example systems, describe what's going on under the hood, and cover the details of network configuration for each of our supported operating systems. In particular, we cover

- Basic configuration
- DHCP client configuration
- Dynamic reconfiguration and tuning
- Security, firewalls, filtering, and NAT configuration
- Quirks

However, not all of our operating systems need discussion for each topic.

Keep in mind that most network configuration happens at boot time, so there's some overlap between the information here and the information presented in Chapter 3, *Booting and Shutting Down*.

14.12 LINUX NETWORKING

Linux is always one of the first networking stacks to include new features. The Linux folks are sometimes so quick that the rest of the networking infrastructure

cannot interoperate. For example, the Linux implementation of explicit congestion notification (ECN), specified in RFC2481, collided with incorrect default settings on an older Cisco firewall product, causing all packets with the ECN bit set to be dropped. Oops.

Linux developers love to tinker, and they often implement features and algorithms that aren't yet accepted standards. One example is the Linux kernel's addition of pluggable congestion control algorithms in release 2.6.13. The several options include variations for lossy networks, high-speed WANs with lots of packet loss, satellite links, and more. The standard TCP "reno" mechanism (slow start, congestion avoidance, fast retransmit, and fast recovery) is still used by default, but a variant may be more appropriate for your environment.

After any change to a file that controls network configuration at boot time, you may need to either reboot or bring the network interface down and then up again for your change to take effect. You can use **ifdown** *interface* and **ifup** *interface* for this purpose on most Linux systems, although the implementations are not identical. (Under SUSE, **ifup** and **ifdown** only work when networking is not under the control of NetworkManager.)

NetworkManager

Linux support for mobile networking was relatively scattershot until the advent of NetworkManager in 2004. It consists of a service that's designed to be run continuously, along with a system tray app for configuring individual network interfaces. In addition to various kinds of wired network, NetworkManager also handles transient wireless networks, wireless broadband, and VPNs. It continually assesses the available networks and shifts service to "preferred" networks as they become available. Wired networks are most preferred, followed by familiar wireless networks.

This system represents quite a change for Linux network configuration. In addition to being more fluid than the traditional static configuration, it's also designed to be run and managed by users rather than system administrators. NetworkManager has been widely adopted by Linux distributions, including all of our examples, but in an effort to avoid breaking existing scripts and setups, it's usually made available as a sort of "parallel universe" of network configuration in addition to whatever traditional network configuration was used in the past.

SUSE makes you choose whether you want to live in the NetworkManager world or use the legacy configuration system, which is managed through YaST. Ubuntu runs NetworkManager by default, but keeps the statically configured network interfaces out of the NetworkManager domain. Red Hat Enterprise Linux doesn't run NetworkManager by default at all.

NetworkManager is primarily of use on laptops, since their network environment may change frequently. For servers and desktop systems, NetworkManager isn't

necessary and may in fact complicate administration. In these environments, it should be ignored or configured out.

Ubuntu network configuration

As shown in Table 14.6, Ubuntu configures the network in **/etc/hostname** and **/etc/network/interfaces**, with a bit of help from the file **/etc/network/options**.

Table 14.6 Ubuntu network configuration files in /etc

File	What's set there
hostname	Hostname
network/interfaces	IP address, netmask, default route

The hostname is set in **/etc/hostname**. The name in this file should be fully qualified; its value is used in a variety of contexts, some of which require qualification.

The IP address, netmask, and default gateway are set in **/etc/network/interfaces**. A line starting with the `iface` keyword introduces each interface. The `iface` line can be followed by indented lines that specify additional parameters. For example:

```
auto lo eth0
iface lo inet loopback
iface eth0 inet static
      address 192.168.1.102
      netmask 255.255.255.0
      gateway 192.168.1.254
```

The **ifup** and **ifdown** commands read this file and bring the interfaces up or down by calling lower-level commands (such as **ifconfig**) with the appropriate parameters. The `auto` clause specifies the interfaces to be brought up at boot time or whenever **ifup -a** is run.

The `inet` keyword in the `iface` line is the address family a la **ifconfig**. The keyword `static` is called a "method" and specifies that the IP address and netmask for eth0 are directly assigned. The `address` and `netmask` lines are required for static configurations; earlier versions of the Linux kernel also required the network address to be specified, but now the kernel is smarter and can figure out the network address from the IP address and netmask. The `gateway` line specifies the address of the default network gateway and is used to install a default route.

SUSE network configuration

SUSE makes you choose between NetworkManager and the traditional configuration system. You make the choice inside of YaST; you can also use the YaST GUI to configure the traditional system. Here, we assume the traditional system. In addition to configuring network interfaces, YaST provides straightforward UIs for the **/etc/hosts** file, static routes, and DNS configuration. Table 14.7 shows the underlying configuration files.

Table 14.7 SUSE network configuration files in /etc/sysconfig/network

File	What's set there
ifcfg-*interface*	Hostname, IP address, netmask, and more
ifroute-*interface*	Interface-specific route definitions
routes	Default route and static routes for all interfaces
config	Lots of less commonly used network variables

With the exceptions of DNS parameters and the system hostname, SUSE sets most networking options in **ifcfg-***interface* files in the **/etc/sysconfig/network** directory. One file should be present for each interface on the system.

In addition to specifying the IP address, gateway, and broadcast information for an interface, the **ifcfg-*** files can tune many other network dials. Take a look at the **ifcfg.template** file for a well-commented rundown of the possible parameters. Here's a simple example with our comments:

```
BOOTPROTO='static'      # Static is implied but it doesn't hurt to be verbose.
IPADDR='192.168.1.4/24' # The /24 defines the NETWORK and NETMASK vars
NAME='AMD PCnet - Fast 79C971' # Used to start and stop the interface.
STARTMODE='auto'        # Start automatically at boot
USERCONTROL='no'        # Disable control through kinternet/cinternet GUI
```

Global static routing information for a SUSE system (including the default route) is stored in the **routes** file. Each line in this file is like a **route** command with the option names omitted and includes destination, gateway, netmask, interface, and optional extra parameters to be stored in the routing table for use by routing daemons. For the host configured above, which has only a default route, the **routes** file contains the line

```
default 192.168.1.254 - -
```

Routes unique to specific interfaces are kept in **ifroute-***interface* files, where the nomenclature of the *interface* component is the same as for the **ifcfg-*** files. The contents have the same format as the **routes** file.

Red Hat network configuration

Red Hat's network configuration GUI is called **system-config-network**; it's also accessible from the System->Administration menu under the name Network. This tool provides a simple UI for configuring individual network interfaces and static routes. It also has panels for setting up IPsec tunnels, configuring DNS, and adding **/etc/hosts** entries.

Table 14.8 shows the underlying configuration files that this GUI edits.

You set the machine's hostname in **/etc/sysconfig/network**, which also contains lines that specify the machine's DNS domain and default gateway.

Table 14.8 Red Hat network configuration files in /etc/sysconfig

File	What's set there
network	Hostname, default route
static-routes	Static routes
network-scripts/ifcfg-*ifname*	Per-interface parameters: IP address, netmask, etc.

For example, here is a **network** file for a host with a single Ethernet interface:

```
NETWORKING=yes
NETWORKING_IPV6=no
HOSTNAME=redhat.toadranch.com
DOMAINNAME=toadranch.com        ### optional
GATEWAY=192.168.1.254
```

Interface-specific data is stored in **/etc/sysconfig/network-scripts/ifcfg**-*ifname*, where *ifname* is the name of the network interface. These configuration files set the IP address, netmask, network, and broadcast address for each interface. They also include a line that specifies whether the interface should be configured "up" at boot time.

A generic machine will have files for an Ethernet interface (eth0) and for the loopback interface (lo). For example,

```
DEVICE=eth0
IPADDR=192.168.1.13
NETMASK=255.255.255.0
NETWORK=192.168.1.0
BROADCAST=192.168.1.255
ONBOOT=yes
```

and

```
DEVICE=lo
IPADDR=127.0.0.1
NETMASK=255.0.0.0
NETWORK=127.0.0.0
BROADCAST=127.255.255.255
ONBOOT=yes
NAME=loopback
```

are the **ifcfg-eth0** and **ifcfg-lo** files for the machine redhat.toadranch.com described in the **network** file above. A DHCP-based setup for eth0 is even simpler:

```
DEVICE=eth0
BOOTPROTO=dhcp
ONBOOT=yes
```

After changing configuration information in **/etc/sysconfig**, run **ifdown** *ifname* followed by **ifup** *ifname* for the appropriate interface. If you reconfigure multiple interfaces at once, you can use the command **service network restart** to reset all

networking. (This is really just a shorthand way to run **/etc/rc.d/init.d/network**, which is invoked at boot time with the **start** argument.)

The startup scripts can also configure static routes. Any routes added to the file **/etc/sysconfig/static-routes** are entered into the routing table at boot time. The entries specify arguments to **route add**, although in a different order:

```
eth0 net 130.225.204.48 netmask 255.255.255.248 gw 130.225.204.49
eth1 net 192.38.8.0 netmask 255.255.255.224 gw 192.38.8.129
```

The interface is specified first, but it is actually shuffled to the end of the **route** command line, where it forces the route to be associated with the given interface. (You'll see this architecture in the GUI as well, where the routes are configured as part of the setup for each interface.) The rest of the line consists of **route** arguments. The **static-routes** example above would produce the following commands:

```
route add -net 130.225.204.48 netmask 255.255.255.248 gw 130.225.204.49 eth0
route add -net 192.38.8.0 netmask 255.255.255.224 gw 192.38.8.129 eth1
```

Current Linux kernels do not use the metric parameter to **route**, but they allow it to be entered into the routing table for use by routing daemons.

Linux network hardware options

The **ethtool** command queries and sets a network interface's media-specific parameters such as link speed and duplex. It replaces the old **mii-tool** command, but some systems still include both.

You can query the status of an interface just by naming it. For example, this eth0 interface (a generic NIC on a PC motherboard) has autonegotiation enabled and is currently running at full speed:

```
ubuntu# ethtool eth0
Settings for eth0:
        Supported ports: [ TP MII ]
        Supported link modes:   10baseT/Half     10baseT/Full
                                100baseT/Half    100baseT/Full
                                1000baseT/Half   1000baseT/Full
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half     10baseT/Full
                                100baseT/Half    100baseT/Full
                                1000baseT/Half   1000baseT/Full
        Advertised auto-negotiation: Yes
        Speed: 1000Mb/s
        Duplex: Full
        Port: MII
        PHYAD: 0
        Transceiver: internal
        Auto-negotiation: on
        Supports Wake-on: pumbg
        Wake-on: g
        Current message level: 0x00000033 (51)
        Link detected: yes
```

To lock this interface to 100 Mb/s full duplex, use the command

```
ubuntu# ethtool -s eth0 speed 100 duplex full
```

If you are trying to determine whether autonegotiation is reliable in your environment, you may also find **ethtool -r** helpful. It forces the parameters of the link to be renegotiated immediately.

Another useful option is **-k**, which shows what protocol-related tasks have been assigned to the network interface rather than being performed by the kernel. Most interfaces can calculate checksums, and some can assist with segmentation as well. Unless you have reason to think that a network interface is not doing these tasks reliably, it's always better to offload them. You can use **ethtool -K** in combination with various suboptions to force or disable specific types of offloading. (The **-k** option shows current values and the **-K** option sets them.)

Any changes you make with **ethtool** are transient. If you want them to be enforced consistently, you'll have to make sure that **ethtool** gets run as part of the system's network configuration. It's best to do this as part of the per-interface configuration; if you just arrange to have some **ethtool** commands run at boot time, your configuration will not properly cover cases in which the interfaces are restarted without a reboot of the system.

On Red Hat systems, you can include an ETHTOOL_OPTS= line in the configuration file for the interface underneath **/etc/sysconfig/network-scripts**. **ifup** passes the entire line as arguments to **ethtool**.

SUSE's provision for running **ethtool** is similar to Red Hat's, but the option is called ETHTOOL_OPTIONS and the per-interface configuration files are kept in **/etc/sysconfig/network**.

In Ubuntu, you can run the **ethtool** commands from a post-up script specified in the interface's configuration in **/etc/network/interfaces**.

Linux TCP/IP options

Linux puts a representation of each tunable kernel variable into the **/proc** virtual filesystem. The networking variables are in **/proc/sys/net/ipv4**. Here's a trimmed list of some of the most interesting ones for illustration:

```
ubuntu$ cd /proc/sys/net/ipv4; ls -F
…                           …                       tcp_no_metrics_save
conf/                       tcp_congestion_control  tcp_orphan_retries
icmp_echo_ignore_all        tcp_dma_copybreak       tcp_reordering
icmp_echo_ignore_broadcasts tcp_dsack               tcp_retrans_collapse
…                           tcp_ecn                 tcp_retries1
icmp_ratelimit              tcp_fack                tcp_retries2
icmp_ratemask               tcp_fin_timeout         tcp_rfc1337
igmp_max_memberships        tcp_frto                tcp_rmem
igmp_max_msf                tcp_frto_response       tcp_sack
inet_peer_gc_maxtime        tcp_keepalive_intvl     ,,,
inet_peer_gc_mintime        tcp_keepalive_probes    tcp_stdurg
```

```
inet_peer_maxttl
inet_peer_minttl
inet_peer_threshold
ip_default_ttl
ip_dynaddr
ip_forward
…
neigh/
route/
tcp_keepalive_time
tcp_low_latency
tcp_max_orphans
tcp_max_ssthresh
tcp_max_syn_backlog
tcp_max_tw_buckets
tcp_mem
tcp_moderate_rcvbuf
tcp_mtu_probing
tcp_synack_retries
tcp_syncookies
tcp_syn_retries
tcp_timestamps
…
udp_mem
udp_rmem_min
udp_wmem_min
```

Many of the variables with rate and max in their names are used to thwart denial of service attacks. The **conf** subdirectory contains variables that are set per interface. It contains subdirectories **all** and **default** and a subdirectory for each interface (including the loopback). Each subdirectory contains the same set of files.

```
ubuntu$ cd conf/default; ls -F
accept_redirects
accept_source_route
arp_accept
arp_announce
arp_filter
arp_ignore
bootp_relay
disable_policy
disable_xfrm
force_igmp_version
forwarding
log_martians
mc_forwarding
medium_id
promote_secondaries
proxy_arp
rp_filter
secure_redirects
send_redirects
shared_media
tag
```

If you change a variable in the **conf/eth0** subdirectory, for example, your change applies to that interface only. If you change the value in the **conf/all** directory, you might expect it to set the corresponding value for all existing interfaces, but this is not in fact what happens. Each variable has its own rules for accepting changes via **all**. Some values are ORed with the current values, some are ANDed, and still others are MAXed or MINed. As far as we are aware, there is no documentation for this process outside of the kernel source code, so the whole debacle is probably best avoided. Just confine your modifications to individual interfaces.

If you change a variable in the **conf/default** directory, the new value propagates to any interfaces that are later configured. On the other hand, it's nice to keep the defaults unmolested as reference information; they make a nice sanity check if you want to undo other changes.

The **/proc/sys/net/ipv4/neigh** directory also contains a subdirectory for each interface. The files in each subdirectory control ARP table management and IPv6 neighbor discovery for that interface. Here is the list of variables; the ones starting with gc (for garbage collection) determine how ARP table entries are timed out and discarded.

```
ubuntu$ cd neigh/default; ls -F
anycast_delay
app_solicit
base_reachable_time
base_reachable_time_ms
delay_first_probe_time
gc_interval
gc_stale_time
gc_thresh1
gc_thresh2
gc_thresh3
locktime
mcast_solicit
proxy_delay
proxy_qlen
retrans_time
retrans_time_ms
ucast_solicit
unres_qlen
```

To see the value of a variable, use **cat**; to set it, use **echo** redirected to the proper filename. For example, the command

```
ubuntu$ cat icmp_echo_ignore_broadcasts
0
```

shows that this variable's value is 0, meaning that broadcast pings are not ignored. To set it to 1 (and avoid falling prey to Smurf-type denial of service attacks), run

```
ubuntu$ sudo sh -c "echo 1 > icmp_echo_ignore_broadcasts"
```
[18]

from the **/proc/sys/net** directory.

You are typically logged in over the same network you are tweaking as you adjust these variables, so be careful! You can mess things up badly enough to require a reboot from the console to recover, which might be inconvenient if the system happens to be in Point Barrow, Alaska, and it's January. Test-tune these variables on your desktop system before you even think of tweaking a production machine.

To change any of these parameters permanently (or more accurately, to reset them every time the system boots), add the appropriate variables to **/etc/sysctl.conf**, which is read by the **sysctl** command at boot time. The format of the **sysctl.conf** file is *variable=value* rather than **echo value > variable** as you would run from the shell to change the variable by hand. Variable names are pathnames relative to **/proc/sys**; you can also use dots instead of slashes if you prefer. For example, either of the lines

```
net.ipv4.ip_forward=0
net/ipv4/ip_forward=0
```

in the **/etc/sysctl.conf** file would turn off IP forwarding on this host.

Some of the options under **/proc** are better documented than others. Your best bet is to look at the man page for the protocol in question in section 7 of the manuals. For example, **man 7 icmp** documents four of the six available options. (You must have man pages for the Linux kernel installed to see man pages about protocols.)

You can also take a look at the **ip-sysctl.txt** file in the kernel source distribution for some good comments. If you don't have kernel source installed, just google for ip-sysctl-txt to reach the same document.

Security-related kernel variables

Table 14.9 shows Linux's default behavior with regard to various touchy network issues. For a brief description of the implications of these behaviors, see page 472. We recommend that you verify the values of these variables so that you do not answer broadcast pings, do not listen to routing redirects, and do not accept

18. If you try this command in the form **sudo echo 1 > icmp_echo_ignore_broadcasts**, you just generate a "permission denied" message—your shell attempts to open the output file before it runs **sudo**. You want the **sudo** to apply to both the **echo** command and the redirection. Ergo, you must create a root subshell in which to execute the entire command.

source-routed packets. These should be the defaults on current distributions except for **accept_redirects** and sometimes **accept_source_route**.

Table 14.9 Default security-related network behaviors in Linux

Feature	Host	Gateway	Control file (in /proc/sys/net/ipv4)
IP forwarding	off	on	**ip_forward** for the whole system **conf/***interface***/forwarding** per interface[a]
ICMP redirects	obeys	ignores	**conf/***interface***/accept_redirects**
Source routing	*varies*	*varies*	**conf/***interface***/accept_source_route**
Broadcast ping	ignores	ignores	**icmp_echo_ignore_broadcasts**

a. The *interface* can be either a specific interface name or **all**.

Linux NAT and packet filtering

Linux traditionally implements only a limited form of Network Address Translation (NAT) that is more properly called Port Address Translation, or PAT. Instead of using a range of IP addresses as a true NAT implementation would, PAT multiplexes all connections onto a single address. The details and differences aren't of much practical importance, though.

iptables implements not only NAT but also packet filtering. In earlier versions of Linux this functionality was a bit of a mess, but **iptables** makes a much cleaner separation between the NAT and filtering features.

Packet filtering features are covered in more detail in the *Security* chapter starting on page 932. If you use NAT to let local hosts access the Internet, you *must* use a full complement of firewall filters when running NAT. The fact that NAT "isn't really IP routing" doesn't make a Linux NAT gateway any more secure than a Linux router. For brevity, we describe only the actual NAT configuration here; however, this is but a small part of a full configuration.

To make NAT work, you must enable IP forwarding in the kernel by setting the **/proc/sys/net/ipv4/ip_forward** kernel variable to 1. Additionally, you must insert the appropriate kernel modules:

```
ubuntu$ sudo /sbin/modprobe iptable_nat
ubuntu$ sudo /sbin/modprobe ip_conntrack
ubuntu$ sudo /sbin/modprobe ip_conntrack_ftp
```

Many other connection-tracking modules exist; see the **net/netfilter** subdirectory underneath **/lib/modules** for a more complete list and enable the ones you need.

The **iptables** command to route packets using NAT is of the form

```
sudo iptables -t nat -A POSTROUTING -o eth1 -j SNAT --to 63.173.189.1
```

In this example, eth0 is the interface connected to the Internet. The eth0 interface does not appear directly in the command line above, but its IP address is the one that appears as the argument to **--to**. The eth1 interface is the one connected to the internal network.

To Internet hosts, it appears that all packets from hosts on the internal network have eth0's IP address. The host performing NAT receives incoming packets, looks up their true destinations, rewrites them with the appropriate internal network IP address, and sends them on their merry way.

14.13 SOLARIS NETWORKING

Solaris comes with a bounteous supply of startup scripts. At a trade show, we once scored a tear-off calendar with sysadmin trivia questions on each day's page. The question for January 1 was to name all the files you had to touch to change the hostname and IP address on a machine running Solaris. A quick peek at the answers showed six files. This is modularization taken to bizarre extremes. That said, let's look at Solaris network configuration.

Solaris basic network configuration

Solaris stashes some network configuration files in **/etc** and some in **/etc/inet**. Many are duplicated through the magic of symbolic links, with the actual files living in **/etc/inet** and the links in **/etc**.

To set the hostname, enter it into the file **/etc/nodename**. The change will take effect when the machine is rebooted. Some sites use just the short hostname; others use the fully qualified domain name.

See page 739 for more information about the name service switch.

The **/etc/defaultdomain** file's name suggests that it might be used to specify the DNS domain, but it actually specifies the NIS or NIS+ domain name. The DNS domain is specified in **/etc/resolv.conf** as usual.

Solaris uses **/etc/nsswitch.conf** to set the order in which **/etc/hosts**, NIS, NIS+, and DNS are consulted for hostname resolution. We recommend looking at the **hosts** file, then DNS for easy booting. The line from **nsswitch.conf** would be

```
hosts:  files  dns
```

This is the default configuration if the host receives the addresses of its DNS servers through DHCP.

Solaris networking can run in traditional mode or in "Network Auto-Magic" (NWAM) mode, where networking is managed autonomously by the **nwamd** daemon. NWAM mode is fine for workstations, but it has limited configurability and allows only one network interface to be active at a time. The discussion below assumes traditional mode.

To see which networking mode is active, run **svcs svc:/network/physical**. There should be two configuration lines, one for NWAM and one for the traditional

21 Network Management and Debugging

Because networks increase the number of interdependencies among machines, they tend to magnify problems. As the saying goes, "Networking is when you can't get any work done because of the failure of a machine you have never heard of."

Network management is the art and science of keeping a network healthy. It generally includes the following tasks:

- Fault detection for networks, gateways, and critical servers
- Schemes for notifying an administrator of problems
- General network monitoring, to balance load and plan expansion
- Documentation and visualization of the network
- Administration of network devices from a central site

On a single network segment, it is generally not worthwhile to establish formal procedures for network management. Just test the network thoroughly after installation and check it occasionally to be sure that its load is not excessive. When it breaks, fix it.

As your network grows, management procedures should become more automated. On a network consisting of several different subnets joined with switches or routers, you may want to start automating management tasks with shell scripts

and simple programs. If you have a WAN or a complex local network, consider installing a dedicated network management station.

In many cases, your organization's reliability needs will dictate the sophistication of your network management system. A problem with the network can bring all work to a standstill. If your site cannot tolerate downtime, it may well be worthwhile to obtain and install a high-end enterprise network management system.

Unfortunately, even the best network management system cannot prevent all failures. It is critical to have a well-documented network and a high-quality staff available to handle the inevitable collapses.

21.1 NETWORK TROUBLESHOOTING

Several good tools are available for debugging a network at the TCP/IP layer. Most give low-level information, so you must understand the main ideas of TCP/IP and routing in order to use the debugging tools.

On the other hand, network issues can also stem from problems with higher-level protocols such as DNS, NFS, and HTTP. You might want to read through Chapter 14, *TCP/IP Networking*, and Chapter 15, *Routing*, before tackling this chapter.

In this section, we start with some general troubleshooting strategy. We then cover several essential tools, including **ping**, **traceroute**, **netstat**, **tcpdump**, and Wireshark. We don't discuss the **arp** command in this chapter, though it, too, is sometimes a useful debugging tool—see page 468 for more information.

Before you attack your network, consider these principles:

- Make one change at a time. Test each change to make sure that it had the effect you intended. Back out any changes that have an undesired effect.
- Document the situation as it was before you got involved, and document every change you make along the way.
- Problems may be transient, so begin by capturing relevant information with tools like **sar** and **nmon**. This information may come in handy as you are unraveling the problem.
- Start at one end of a system or network and work through the system's critical components until you reach the problem. For example, you might start by looking at the network configuration on a client, work your way up to the physical connections, investigate the network hardware, and finally, check the server's physical connections and software configuration.
- Communicate regularly. Most network problems affect lots of different people: users, ISPs, system administrators, telco engineers, network administrators, etc. Clear, consistent communication prevents you from hindering one another's efforts to solve the problem.

- Work as a team. Years of experience show that people make fewer stupid mistakes if they have a peer helping out. If the problem has any visibility, management will also want to be involved. Take advantage of managers' interest to get technical people from other groups on board and to cut through red tape where necessary.
- Use the layers of the network to negotiate the problem. Start at the "top" or "bottom" and work your way through the protocol stack.

This last point deserves a bit more discussion. As described on page 450, the architecture of TCP/IP defines several layers of abstraction at which components of the network can function. For example, HTTP depends on TCP, TCP depends on IP, IP depends on the Ethernet protocol, and the Ethernet protocol depends on the integrity of the network cable. You can dramatically reduce the amount of time spent debugging a problem if you first figure out which layer is misbehaving.

Ask yourself questions like these as you work up or down the stack:

- Do you have physical connectivity and a link light?
- Is your interface configured properly?
- Do your ARP tables show other hosts?
- Is there a firewall on your local machine?
- Is there a firewall anywhere between you and the destination?
- If firewalls are involved, do they pass ICMP ping packets and responses?
- Can you ping the localhost address (127.0.0.1)?
- Can you ping other local hosts by IP address?
- Is DNS working properly?[1]
- Can you ping other local hosts by hostname?
- Can you ping hosts on another network?
- Do high-level services such as web and SSH servers work?
- Did you really check the firewalls?

Once you've identified where the problem lies and have a fix in mind, take a step back to consider the effect that your subsequent tests and prospective fixes will have on other services and hosts.

21.2 PING: CHECK TO SEE IF A HOST IS ALIVE

The **ping** command is embarrassingly simple, but in many situations it is the only command you need for network debugging. It sends an ICMP ECHO_REQUEST packet to a target host and waits to see if the host answers back.

1. If a machine hangs at boot time, boots very slowly, or hangs on inbound SSH connections, DNS should be a prime suspect. Solaris and Linux use a sophisticated approach to name resolution that's configurable in **/etc/nsswitch.conf**. On these systems, the name service caching daemon (**nscd**) is of particular interest. If it crashes or is misconfigured, name lookups are affected. With the transition to IPv6 progressing, we find that many DSL routers provide DNS forwarding services that simply drop requests for IPv6 (AAAA) DNS records. This "optimization" causes long timeouts on all name resolution requests. Use the **getent** command to check whether your resolver and name servers are working properly (e.g., **getent hosts google.com**).

You can use **ping** to check the status of individual hosts and to test segments of the network. Routing tables, physical networks, and gateways are all involved in processing a ping, so the network must be more or less working for **ping** to succeed. If **ping** doesn't work, you can be pretty sure that nothing more sophisticated will work either.

However, this rule does not apply to networks that block ICMP echo requests with a firewall. Make sure that a firewall isn't interfering with your debugging before you conclude that the target host is ignoring a **ping**. You might consider disabling a meddlesome firewall for a short period of time to facilitate debugging.

If your network is in bad shape, chances are that DNS is not working. Simplify the situation by using numeric IP addresses when pinging, and use **ping**'s **-n** option to prevent **ping** from attempting to do reverse lookups on IP addresses—these lookups also trigger DNS requests.

Be aware of the firewall issue if you're using **ping** to check your Internet connectivity, too. Some well-known sites answer **ping** packets and others don't. We've found google.com to be a consistent responder.

Most versions of **ping** run in an infinite loop unless you supply a packet count argument. Under Solaris, **ping -s** provides the extended output that other versions use by default. Once you've had your fill of pinging, type the interrupt character (usually <Control-C>) to get out.

Here's an example:

```
linux$ ping beast
PING beast (10.1.1.46): 56 bytes of data.
64 bytes from beast (10.1.1.46): icmp_seq=0 ttl=54 time=48.3ms
64 bytes from beast (10.1.1.46): icmp_seq=1 ttl=54 time=46.4ms
64 bytes from beast (10.1.1.46): icmp_seq=2 ttl=54 time=88.7ms
^C
--- beast ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2026ms
rtt min/avg/max/mdev = 46.490/61.202/88.731/19.481 ms
```

The output for beast shows the host's IP address, the ICMP sequence number of each response packet, and the round trip travel time. The most obvious thing that the output above tells you is that the server beast is alive and connected to the network.

The ICMP sequence number is a particularly valuable piece of information. Discontinuities in the sequence indicate dropped packets. They're normally accompanied by a message for each missing packet.

Despite the fact that IP does not guarantee the delivery of packets, a healthy network should drop very few of them. Lost-packet problems are important to track down because they tend to be masked by higher-level protocols. The network may appear to function correctly, but it will be slower than it ought to be, not only

because of the retransmitted packets but also because of the protocol overhead needed to detect and manage them.

To track down the cause of disappearing packets, first run **traceroute** (see the next section) to discover the route that packets are taking to the target host. Then ping the intermediate gateways in sequence to discover which link is dropping packets. To pin down the problem, you need to send a fair number of packets. The fault generally lies on the link between the last gateway you can ping without loss of packets and the gateway beyond that.

The round trip time reported by **ping** gives you insight into the overall performance of a path through a network. Moderate variations in round trip time do not usually indicate problems. Packets may occasionally be delayed by tens or hundreds of milliseconds for no apparent reason; that's just the way IP works. You should expect to see a fairly consistent round trip time for the majority of packets, with occasional lapses. Many of today's routers implement rate-limited or lower-priority responses to ICMP packets, which means that a router may delay responding to your ping if it is already dealing with a lot of other traffic.

The **ping** program can send echo request packets of any size, so by using a packet larger than the MTU of the network (1,500 bytes for Ethernet), you can force fragmentation. This practice helps you identify media errors or other low-level issues such as problems with a congested network or VPN.

On Linux systems, you specify the desired packet size in bytes with the **-s** flag.

```
$ ping -s 1500 cuinfo.cornell.edu
```

Under Solaris, HP-UX, and AIX, you simply add the desired packet size to the end of the **ping** command.

```
$ ping cuinfo.cornell.edu 1500
```

Note that even a simple command like **ping** can have dramatic effects. In 1998, the so-called Ping of Death attack crashed large numbers of UNIX and Windows systems. It was launched simply by transmission of an overly large ping packet. When the fragmented packet was reassembled, it filled the receiver's memory buffer and crashed the machine. The Ping of Death issue has long since been fixed, but several other caveats should be kept in mind regarding **ping**.

First, it is hard to distinguish the failure of a network from the failure of a server with only the **ping** command. In an environment where ping tests normally work, a failed ping just tells you that *something* is wrong.

Second, a successful ping does not guarantee much about the target machine's state. Echo request packets are handled within the IP protocol stack and do not require a server process to be running on the probed host. A response guarantees only that a machine is powered on and has not experienced a kernel panic. You'll need higher-level methods to verify the availability of individual services such as HTTP and DNS.

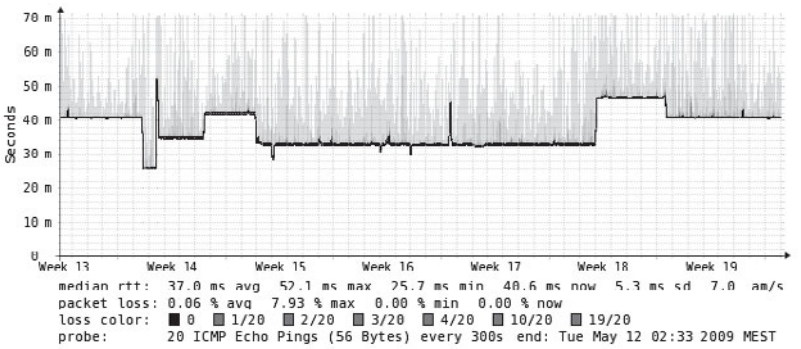
21.3 SMOKEPING: GATHER PING STATISTICS OVER TIME

As mentioned earlier, even a healthy network occasionally drops a packet. On the other hand, networks should not drop packets regularly, even at a low rate, because the impact on users can be disproportionately severe. Because high-level protocols often function even in the presence of packet loss, you might never notice dropped packets unless you're actively monitoring for them.

For this purpose, you may want to check out SmokePing, an open source tool by Tobias Oetiker that keeps track of network latencies. SmokePing sends several ping packets to a target host at regular intervals. It shows the history of each monitored link through a web front end and can send alarms when things go amiss. You can get a copy from oss.oetiker.ch/smokeping.

Exhibit A shows a SmokePing graph. The vertical axis is the round trip time of pings, and the horizontal axis is time (weeks). The black line from which the gray spikes stick up indicates the median round trip time; the spikes themselves are the transit times of individual packets. Since the gray in this graph appears only above the median line, the great majority of packets must be traveling at close to the median speed, with just a few being delayed. This is a typical finding.

Exhibit A Sample SmokePing graph



The stair-stepped shape of the median line indicates that the baseline transit time to this destination has changed several times during the monitoring period. The most likely hypotheses to explain this observation are either that the host is reachable by several routes or that it is actually a collection of several hosts that have the same DNS name but multiple IP addresses.

21.4 TRACEROUTE: TRACE IP PACKETS

traceroute, originally written by Van Jacobson, uncovers the sequence of gateways through which an IP packet travels to reach its destination. All modern operating systems come with some version of **traceroute**.[2] The syntax is simply

traceroute *hostname*

There are a variety of options, most of which are not important in daily use. As usual, the *hostname* can be specified as either a DNS name or an IP address. The output is simply a list of hosts, starting with the first gateway and ending at the destination. For example, a **traceroute** from our host jaguar to our host nubark produces the following output:

```
$ traceroute nubark
traceroute to nubark (192.168.2.10), 30 hops max, 38 byte packets
 1  lab-gw (172.16.8.254)  0.840 ms  0.693 ms  0.671 ms
 2  dmz-gw (192.168.1.254)  4.642 ms  4.582 ms  4.674 ms
 3  nubark (192.168.2.10)  7.959 ms  5.949 ms  5.908 ms
```

From this output we can tell that jaguar is three hops away from nubark, and we can see which gateways are involved in the connection. The round trip time for each gateway is also shown—three samples for each hop are measured and displayed. A typical **traceroute** between Internet hosts often includes more than 15 hops, even if the two sites are just across town.

traceroute works by setting the time-to-live field (TTL, actually "hop count to live") of an outbound packet to an artificially low number. As packets arrive at a gateway, their TTL is decreased. When a gateway decreases the TTL to 0, it discards the packet and sends an ICMP "time exceeded" message back to the originating host.

See page 582 for more information about reverse DNS lookups.

The first three **traceroute** packets have their TTL set to 1. The first gateway to see such a packet (lab-gw in this case) determines that the TTL has been exceeded and notifies jaguar of the dropped packet by sending back an ICMP message. The sender's IP address in the header of the error packet identifies the gateway, and **traceroute** looks up this address in DNS to find the gateway's hostname.

To identify the second-hop gateway, **traceroute** sends out a second round of packets with TTL fields set to 2. The first gateway routes the packets and decreases their TTL by 1. At the second gateway, the packets are then dropped and ICMP error messages are generated as before. This process continues until the TTL is equal to the number of hops to the destination host and the packets reach their destination successfully.

2. Even Windows has it, but the command is spelled **tracert** (extra history points if you can guess why).

Most routers send their ICMP messages from the interface "closest" to the destination. If you run **traceroute** backward from the destination host, you may see different IP addresses being used to identify the same set of routers. You might also discover that packets flowing in the reverse direction take a completely different path, a configuration known as asymmetric routing.

Since **traceroute** sends three packets for each value of the TTL field, you may sometimes observe an interesting artifact. If an intervening gateway multiplexes traffic across several routes, the packets might be returned by different hosts; in this case, **traceroute** simply prints them all.

Let's look at a more interesting example from a host in Switzerland to caida.org at the San Diego Supercomputer Center:[3]

```
linux$ traceroute caida.org
traceroute to caida.org (192.172.226.78), 30 hops max, 46 byte packets
 1  gw-oetiker.init7.net (213.144.138.193)  1.122 ms  0.182 ms  0.170 ms
 2  r1zur1.core.init7.net (77.109.128.209)  0.527 ms  0.204 ms  0.202 ms
 3  r1fra1.core.init7.net (77.109.128.250)  18.279 ms  6.992 ms  16.597 ms
 4  r1ams1.core.init7.net (77.109.128.154)  19.549 ms  21.855 ms  13.514 ms
 5  r1lon1.core.init7.net (77.109.128.150)  19.165 ms  21.157 ms  24.866 ms
 6  r1lax1.ce.init7.net (82.197.168.69)  158.232 ms  158.224 ms  158.271 ms
 7  cenic.laap.net (198.32.146.32)  158.349 ms  158.309 ms  158.248 ms
 8  dc-lax-core2--lax-peer1-ge.cenic.net (137.164.46.119)  158.60 ms * 158.71 ms
 9  dc-tus-agg1--lax-core2-10ge.cenic.net (137.164.46.7) 159 ms 159 ms 159 ms
10  dc-sdsc-sdsc2--tus-dc-ge.cenic.net (137.164.24.174) 161 ms 161 ms 161 ms
11  pinot.sdsc.edu (198.17.46.56)  161.559 ms  161.381 ms  161.439 ms
12  rommie.caida.org (192.172.226.78)  161.442 ms  161.445 ms  161.532 ms
```

This output shows that packets travel inside Init Seven's network for a long time. Sometimes we can guess the location of the gateways from their names. Init Seven's core stretches all the way from Zurich (zur) to Frankfurt (fra), Amsterdam (ams), London (lon), and finally, Los Angeles (lax). Here, the traffic transfers to cenic.net, which delivers the packets to the caida.org host within the network of the San Diego Supercomputer Center (sdsc) in La Jolla, CA.

At hop 8, we see a star in place of one of the round trip times. This notation means that no response (error packet) was received in response to the probe. In this case, the cause is probably congestion, but that is not the only possibility. **traceroute** relies on low-priority ICMP packets, which many routers are smart enough to drop in preference to "real" traffic. A few stars shouldn't send you into a panic.

If you see stars in all the time fields for a given gateway, no "time exceeded" messages are arriving from that machine. Perhaps the gateway is simply down. Sometimes, a gateway or firewall is configured to silently discard packets with expired TTLs. In this case, you can still see through the silent host to the gateways beyond. Another possibility is that the gateway's error packets are slow to return and that **traceroute** has stopped waiting for them by the time they arrive.

3. We removed a few fractions of milliseconds from the longer lines to keep them from folding.

Some firewalls block ICMP "time exceeded" messages entirely. If such a firewall lies along the path, you won't get information about any of the gateways beyond it. However, you can still determine the total number of hops to the destination because the probe packets eventually get all the way there.

Also, some firewalls may block the outbound UDP datagrams that **traceroute** sends to trigger the ICMP responses. This problem causes **traceroute** to report no useful information at all. If you find that your own firewall is preventing you from running **traceroute**, make sure the firewall has been configured to pass UDP ports 33434–33534 as well as ICMP ECHO (type 8) packets.

A slow link does not necessarily indicate a malfunction. Some physical networks have a naturally high latency; UMTS/EDGE/GPRS wireless networks are a good example. Sluggishness can also be a sign of high load on the receiving network. Inconsistent round trip times would support such a hypothesis.

Sometimes, you may see the notation `!N` instead of a star or round trip time. It indicates that the current gateway sent back a "network unreachable" error, meaning that it doesn't know how to route your packet. Other possibilities include `!H` for "host unreachable" and `!P` for "protocol unreachable." A gateway that returns any of these error messages is usually the last hop you can get to. That host usually has a routing problem (possibly caused by a broken network link): either its static routes are wrong, or dynamic protocols have failed to propagate a usable route to the destination.

If **traceroute** doesn't seem to be working for you or is working slowly, it may be timing out while trying to resolve the hostnames of gateways through DNS. If DNS is broken on the host you are tracing from, use **traceroute -n** to request numeric output. This option disables hostname lookups; it may be the only way to get **traceroute** to function on a crippled network.

traceroute needs root privileges to operate. To be available to normal users, it must be installed setuid root. Several Linux distributions include the **traceroute** command but leave off the setuid bit. Depending on your environment and needs, you can either turn the setuid bit back on or give interested users access to the command through **sudo**.

Recent years have seen the introduction of several new **traceroute**-like utilities that can bypass ICMP-blocking firewalls. See the PERTKB Wiki for an overview of these tools at tinyurl.com/y99qh6u. We especially like **mtr**, which has a **top**-like interface and shows a sort of live **traceroute**. Very neat!

When debugging routing issues, it can be helpful to take a look at your site from the perspective of the outside world. Several web-based route tracing services let you do this sort of inverse **traceroute** right from a browser window. Thomas Kernen maintains a list of these services at traceroute.org.

21.5 NETSTAT: GET NETWORK STATISTICS

netstat collects a wealth of information about the state of your computer's networking software, including interface statistics, routing information, and connection tables. There isn't really a unifying theme to the different sets of output, except that they all relate to the network. Think of **netstat** as the "kitchen sink" of network tools—it exposes a variety of network information that doesn't fit anywhere else. Here, we discuss the five most common uses of **netstat**:

- Inspecting interface configuration information
- Monitoring the status of network connections
- Identifying listening network services
- Examining the routing table
- Viewing operational statistics for various network protocols

Inspecting interface configuration information

netstat -i shows the configuration and state of each of the host's network interfaces along with the associated traffic counters. The output is generally tabular but the details vary by system:

```
solaris$ netstat -i
Name    Mtu  Net/Dest  Address   Ipkts      Ierrs Opkts      Oerrs Collis Queue
lo0     8232 loopback  localhost 319589661  0     319589661  0     0      0
e1000g1 1500 host-if1  host-if1  369842112  0     348557584  0     0      0
e1000g2 1500 host-if2  host-if2  93141891   0     121107161  0     0      0
```

```
hp-ux$ netstat -i
Name Mtu    Network      Address            Ipkts    Ierrs Opkts    Oerrs Coll
lan0 1500   192.168.10.0 hpux11             2611259  0     2609847  0     0
lo0  32808  loopback     hpux11.atrust.com
```

```
aix$ netstat -i
Name Mtu    Network    Address          ZoneID Ipkts    Ierrs Opkts    Oerrs Coll
en3  1500   link#2     0.11.25.39.e0.b6       41332    0     14173    3     0
en3  1500   192.168.10 IBM                    41332    0     14173    3     0
lo0  16896  link#1                           1145121  0     1087387  0     0
lo0  16896  127        loopback              1145121  0     1087387  0     0
lo0  16896  ::1                           0  1145121  0     1087387  0     0
```

On Linux, you may want to use **ifconfig -a** instead of **netstat -i**. It prints similar information in a more detailed and more verbose format.

```
linux$ ifconfig -a
eth0      Link encap:EthernetHWaddr 00:15:17:4c:4d:00
          inet addr:192.168.0.203Bcast:192.168.0.255Mask:255.255.255.0
          inet6 addr: fe80::215:17ff:fe4c:4d00/64 Scope:Link
          UP BROADCAST RUNNING MULTICASTMTU:1500Metric:1
          RX packets:559543852 errors:0 dropped:62 overruns:0 frame:0
          TX packets:457050867 errors:0 dropped:0 overruns:0 carrier:0
```

```
          collisions:0 txqueuelen:1000
          RX bytes:478438325085 (478.4 GB)TX bytes:228502292340 (228.5 GB)
          Memory:b8820000-b8840000

eth1      Link encap:EthernetHWaddr 00:15:17:4c:4d:01
          BROADCAST MULTICASTMTU:1500Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)TX bytes:0 (0.0 B)
          Memory:b8800000-b8820000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNINGMTU:16436Metric:1
          RX packets:1441988 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1441988 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:327048609 (327.0 MB)TX bytes:327048609 (327.0 MB)
```

This host has two network interfaces: one for regular traffic, plus a second interface that is currently not in use (it has no IP address and is not marked `UP`). `RX packets` and `TX packets` report the number of packets that have been received and transmitted on each interface since the machine was booted. Many types of errors are counted in the error buckets, and it is normal for a few to show up.

Errors should be less than 1% of the associated packets. If your error rate is high, compare the rates of several neighboring machines. A large number of errors on a single machine suggests a problem with that machine's interface or connection. A high error rate everywhere most likely indicates a media or network problem. One of the most common causes of a high error rate is an Ethernet speed or duplex mismatch caused by a failure of autosensing or autonegotiation.

Although a collision is a type of error, it is counted separately by **netstat**. The field labeled `Collisions` reports the number of collisions that were experienced while packets were being sent. Use this number to calculate the percentage of output packets (`TX packets`) that resulted in collisions.

On a switched network with full duplex links—that is, on any modern variety of Ethernet—you should not see any collisions, even when the network is under heavy load. If you do see collisions, something is seriously wrong. You might also want to make sure that flow control is enabled on your switches and routers, especially if your network contains links of different speeds.

We have often traced network problems back to el cheapo pieces of desktop network equipment, such as a switch that has gone haywire and needs to be power-cycled or replaced.

Monitoring the status of network connections

With no arguments, **netstat** displays the status of active TCP and UDP ports. Inactive ("listening") servers that are waiting for connections are normally hidden, but you can see them with **netstat -a**.[4] The output looks like this:

```
linux$ netstat -a
Active Internet connections (servers and established)
Proto    Recv-Q  Send-Q  Local Address   ForeignAddress      State
tcp      0       0       *:ldap          *:*                 LISTEN
tcp      0       0       *:mysql         *:*                 LISTEN
tcp      0       0       *:imaps         *:*                 LISTEN
tcp      0       0       bull:ssh        dhcp-32hw:4208      ESTABLISHED
tcp      0       0       bull:imaps      nubark:54195        ESTABLISHED
tcp      0       0       bull:http       dhcp-30hw:2563      ESTABLISHED
tcp      0       0       bull:imaps      dhcp-18hw:2851      ESTABLISHED
tcp      0       0       *:http          *:*                 LISTEN
tcp      0       0       bull:37203      baikal:mysql        ESTABLISHED
tcp      0       0       *:ssh           *:*                 LISTEN
…
```

This example is from the host otter, and it has been severely pruned; for example, UDP and UNIX socket connections are not displayed. The output above shows an inbound SSH connection, two inbound IMAPS connections, one inbound HTTP connection, an outbound MySQL connection, and a bunch of ports listening for other connections.

Addresses are shown as *hostname.service*, where the *service* is a port number. For well-known services, **netstat** shows the port symbolically, using the mapping defined in **/etc/services**. You can obtain numeric addresses and ports with the **-n** option to **netstat**. Like most network debugging tools, **netstat** is painful to use without the **-n** flag if your DNS is broken.

`Send-Q` and `Recv-Q` show the sizes of the local host's send and receive queues for the connection; the queue sizes on the other end of a TCP connection might be different. These numbers should tend toward 0 and at least not be consistently nonzero. (Of course, if you are running **netstat** over a network terminal, the send queue for your connection may never be 0.)

The connection state has meaning only for TCP; UDP is a connectionless protocol. The most common states you'll see are `ESTABLISHED` for currently active connections, `LISTEN` for servers waiting for connections (not normally shown without **-a**), and `TIME_WAIT` for connections in the process of closing.

netstat -a is primarily useful for debugging higher-level problems once you have determined that basic networking facilities are working correctly. It lets you verify that servers are set up correctly and facilitates the diagnosis of certain types of miscommunication, particularly with TCP. For example, a connection that stays

4. Connections for "UNIX domain sockets" are also shown, but since they aren't related to networking, we do not discuss them here.

in state SYN_SENT identifies a process that is trying to contact a nonexistent or inaccessible network server.

See Chapter 13 for more information about kernel tuning.

If **netstat** shows a lot of connections in the SYN_WAIT condition, your host probably cannot handle the number of connections being requested. This inadequacy may be due to kernel tuning limitations or even to malicious flooding.

Identifying listening network services

One common question in this security-conscious era is "What processes on this machine are listening on the network for incoming connections?" **netstat -a** shows all the ports that are actively listening (any TCP port in state LISTEN, and potentially any UDP port), but on a busy machine, those lines can get lost in the noise of established TCP connections.

On Linux, use **netstat -l** to see only the listening ports. The output format is the same as for **netstat -a**. You can also add the **-p** flag to make **netstat** identify the specific process associated with each listening port.[5] The sample output below shows three common services (**sshd**, **sendmail**, and **named**) followed by an unusual one:

```
linux$ netstat -lp
…
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      23858/sshd
tcp        0      0 0.0.0.0:25              0.0.0.0:*               LISTEN      10342/sendmail
udp        0      0 0.0.0.0:53              0.0.0.0:*                           30016/named
udp        0      0 0.0.0.0:962             0.0.0.0:*                           38221/mudd
…
```

mudd with PID 38221 is listening on UDP port 962. If you don't know what **mudd** is, you might want to follow up on this.

For security, it's also helpful to look at machines from an outsider's perspective by running a port scanner. **nmap** is very helpful for this; see page 914.

Examining the routing table

netstat -r displays the kernel's routing table. The following sample output is from a Red Hat machine with two network interfaces. (The output varies slightly among operating systems.)

```
redhat$ netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags MSS Window  irtt Iface
192.168.1.0     0.0.0.0         255.255.255.0   U       0 0          0 eth0
10.2.5.0        0.0.0.0         255.255.255.0   U       0 0          0 eth1
127.0.0.0       0.0.0.0         255.0.0.0       U       0 0          0 lo
0.0.0.0         192.168.1.254   0.0.0.0         UG      0 0         40 eth0
…
```

5. On UNIX systems that don't support **netstat**'s **-p** flag, the **lsof** command can provide this information (and more). See page 145 for more about **lsof**.

Destinations and gateways can be displayed either as hostnames or as IP addresses; the **-n** flag requests numeric output.

See page 466 for more information about the routing table.

Flags characterize the route: U means up (active), G is a gateway, and H is a host route. U, G, and H together indicate a host route that passes through an intermediate gateway. The D flag (not shown) indicates a route resulting from an ICMP redirect. The remaining fields show TCP segment and window sizes along this route along with an initial round trip time estimate and the name of the interface.

Use this form of **netstat** to check the health of your system's routing table. It's particularly important to verify that the system has a default route and that this route is correct. The default route is represented by an all-0 destination address (0.0.0.0) or by the word default. It is possible not to have a default route entry, but such a configuration would be highly atypical on anything but a backbone router.

Viewing operational statistics for network protocols

netstat -s dumps the contents of counters that are scattered throughout the network code. The output has separate sections for IP, ICMP, TCP, and UDP. Below are pieces of **netstat -s** output from a typical server; they have been edited to show only the tastiest pieces of information.

```
Ip:
    671349985 total packets received
    0 forwarded
    345 incoming packets discarded
    667912993 incoming packets delivered
    589623972 requests sent out
    60 dropped because of missing route
    203 fragments dropped after timeout
```

Be sure to check that packets are not being dropped or discarded. It is acceptable for a few incoming packets to be discarded, but a quick rise in this metric usually indicates a memory shortage or some other resource problem.

```
Icmp:
    242023 ICMP messages received
    912 input ICMP message failed.
    ICMP input histogram:
        destination unreachable: 72120
        timeout in transit: 573
        echo requests: 17135
        echo replies: 152195
    66049 ICMP messages sent
    0 ICMP messages failed
    ICMP output histogram:
        destination unreachable: 48914
        echo replies: 17135
```

In this example, the number of echo requests in the input section matches the number of echo replies in the output section. Note that "destination unreachable" messages can still be generated even when all packets are apparently forwardable.

Bad packets eventually reach a gateway that rejects them, and error messages are then sent back along the gateway chain.

```
Tcp:
    4442780 active connections openings
    1023086 passive connection openings
    50399 failed connection attempts
    0 connection resets received
    44 connections established
    666674854 segments received
    585111784 segments send out
    107368 segments retransmited
    86 bad segments received.
    3047240 resets sent
Udp:
    4395827 packets received
    31586 packets to unknown port received.
    0 packet receive errors
    4289260 packets sent
```

It's a good idea to develop a feel for the normal ranges of these statistics so that you can recognize pathological states.

21.6 INSPECTION OF LIVE INTERFACE ACTIVITY

One good way to identify network problems is to look at what's happening right now. How many packets were sent in the last five minutes on a given interface? How many bytes? Are collisions or other errors occurring? You can answer all these questions by watching network activity in real time. Different tools come into play depending on your OS.

On Solaris, you can append an interval in seconds and a count value to **netstat -i**:

```
solaris$ netstat -i 2 3
     input   e1000g    output       input   (Total)     output
packets  errs  packets errs  colls  packets  errs  packets  errs  colls
17861    0     26208   0     0      17951    0     26298    0     0
4        0     2       0     0      4        0     2        0     0
1        0     1       0     0      1        0     1        0     0
…
```

HP-UX and AIX expect a single number that sets the interval (in seconds) at which statistics are to be printed.

```
$ netstat -i 2
   (lan0)-> input          output       (Total)-> input          output
          packets         packets              packets         packets
          9053713         9052513              10115002        10113803
                8               8                     8               8
               22              22                    22              22
                9               9                     9               9
…
```

Linux's **netstat** has no interval option, so for Linux we recommend a completely different tool: **sar**. (We discuss **sar** from the perspective of general system monitoring on page 1129.) Most distributions don't install **sar** by default, but it's always available as an optional package. The example below requests reports every two seconds for a period of one minute (i.e., 30 reports). The **DEV** argument is a literal keyword, not a placeholder for a device or interface name.

```
redhat$ sar -n DEV 2 30
17:50:43 IFACE rxpck/s txpck/s  rxbyt/s  txbyt/s rxcmp/s txcmp/s rxmcst/s
17:50:45    lo    3.61    3.61   263.40   263.40    0.00    0.00    0.00
17:50:45  eth0   18.56   11.86  1364.43  1494.33    0.00    0.00    0.52
17:50:45  eth1    0.00    0.00     0.00     0.00    0.00    0.00    0.00
```

This example is from a Red Hat machine with two network interfaces. The output includes instantaneous and average readings of interface utilization in units of both bytes and packets. The second interface (eth1) is clearly not in use.

The first two columns state the time at which the data was sampled and the names of the network interfaces. The next two columns show the number of packets received and transmitted, respectively.

The rxbyt/s and txbyt/s columns are probably the most useful since they show the actual bandwidth in use. The final three columns give statistics on compressed (rxcmp/s, txcmp/s) and multicast (rxmcst/s) packets.

sar -n DEV is especially useful for tracking down the source of errors. **ifconfig** can alert you to the existence of problems, but it can't tell you whether the errors came from a continuous, low-level problem or from a brief but catastrophic event. Observe the network over time and under a variety of load conditions to solidify your impression of what's going on. Try running **ping** with a large packet payload (size) while you watch the output of **sar -n DEV**.

21.7 PACKET SNIFFERS

tcpdump and Wireshark belong to a class of tools known as packet sniffers. They listen to network traffic and record or print packets that meet criteria of your choice. For example, you can inspect all packets sent to or from a particular host, or TCP packets related to one particular network connection.

Packet sniffers are useful both for solving problems that you know about and for discovering entirely new problems. It's a good idea to take an occasional sniff of your network to make sure the traffic is in order.

Packet sniffers need to be able to intercept traffic that the local machine would not normally receive (or at least, pay attention to), so the underlying network hardware must allow access to every packet. Broadcast technologies such as Ethernet work fine, as do most other modern local area networks.

See page 537 for more information about network switches.

Since packet sniffers need to see as much of the raw network traffic as possible, they can be thwarted by network switches, which by design try to limit the propagation of "unnecessary" packets. However, it can still be informative to try out a sniffer on a switched network. You may discover problems related to broadcast or multicast packets. Depending on your switch vendor, you may be surprised at how much traffic you can see.

In addition to having access to all network packets, the interface hardware must transport those packets up to the software layer. Packet addresses are normally checked in hardware, and only broadcast/multicast packets and those addressed to the local host are relayed to the kernel. In "promiscuous mode," an interface lets the kernel read all packets on the network, even the ones intended for other hosts.

Packet sniffers understand many of the packet formats used by standard network services, and they can print these packets in human-readable form. This capability makes it easier to track the flow of a conversation between two programs. Some sniffers print the ASCII contents of a packet in addition to the packet header and so are useful for investigating high-level protocols.

Since some protocols send information (and even passwords) across the network as cleartext, you must take care not to invade the privacy of your users. On the other hand, nothing quite dramatizes the need for cryptographic security like the sight of a plaintext password captured in a network packet.

Sniffers read data from a raw network device, so they must run as root. Although this root limitation serves to decrease the chance that normal users will listen in on your network traffic, it is really not much of a barrier. Some sites choose to remove sniffer programs from most hosts to reduce the chance of abuse. If nothing else, you should check your systems' interfaces to be sure they are not running in promiscuous mode without your knowledge or consent. On all systems, the output of **ifconfig** labels promiscuous interfaces with the flag PROMISC. On Linux systems, the fact that an interface has been switched to promiscuous mode is also recorded in the kernel log.

tcpdump: industry-standard packet sniffer

tcpdump, yet another amazing network tool by Van Jacobson, is available as a package for most Linux distributions and can be installed from source on our other example systems. **tcpdump** has long been the industry-standard sniffer, and most other network analysis tools read and write trace files in **tcpdump** format, also known as **libpcap** format.

By default, **tcpdump** tunes in on the first network interface it comes across. If it chooses the wrong interface, you can force an interface with the **-i** flag. If DNS is broken or you just don't want **tcpdump** doing name lookups, use the **-n** option. This option is important because slow DNS service can cause the filter to start dropping packets before they can be dealt with by **tcpdump**.

The **-v** flag increases the information you see about packets, and **-vv** gives you even more data. Finally, **tcpdump** can store packets to a file with the **-w** flag and can read them back in with the **-r** flag.

Note that **tcpdump -w** saves only packet headers by default. This default makes for small dumps, but the most helpful and relevant information may be missing. So, unless you are sure you need only headers, use the **-s** option with a value on the order of 1560 (actual values are MTU-dependent) to capture whole packets for later inspection.

As an example, the following truncated output comes from the machine named nubark. The filter specification **host bull** limits the display of packets to those that directly involve the machine bull, either as source or as destination.

```
$ sudo tcpdump host bull
12:35:23.519339 bull.41537 > nubark.domain:  A? atrust.com. (28) (DF)
12:35:23.519961 nubark.domain > bull.41537:  A 66.77.122.161 (112) (DF)
```

The first packet shows bull sending a DNS lookup request about atrust.com to nubark. The response is the IP address of the machine associated with that name, which is 66.77.122.161. Note the time stamp on the left and **tcpdump**'s understanding of the application-layer protocol (in this case, DNS). The port number on bull is arbitrary and is shown numerically (41537), but since the server port number (53) is well known, **tcpdump** shows its symbolic name, domain.

Packet sniffers can produce an overwhelming amount of information—overwhelming not only for you but also for the underlying operating system. To avoid this problem on busy networks, **tcpdump** lets you specify complex filters. For example, the following filter collects only incoming web traffic from one subnet:

```
$ sudo tcpdump src net 192.168.1.0/24 and dst port 80
```

The **tcpdump** man page contains several good examples of advanced filtering along with a complete listing of primitives.[6]

Solaris includes a sniffer in the base system that works much like **tcpdump**. It is called **snoop**. HP-UX, AIX, and most Linux distributions do not seem to include a packet sniffer in the base install.

```
solaris$ snoop
Using device /dev/e1000g0 (promiscuous mode)
nubark -> solaris TCP D=22 S=58689 Ack=2141650294 Seq=3569652094 Len=0
      Win=15008 Options=<nop,nop,tstamp 292567745 289381342>
nubark -> solaris TCP D=22 S=58689 Ack=2141650358 Seq=3569652094 Len=0
      Win=15008 Options=<nop,nop,tstamp 292567745 289381342>
? -> (multicast)  ETHER Type=023C (LLC/802.3), size = 53 bytes
…
```

6. If your filtering needs exceed **tcpdump**'s capabilities, consider **ngrep** (ngrep.sourceforge.net), which can filter packets according to their contents.

If you are using Solaris zones, note that **snoop** only works properly in the global zone, even when you are debugging a problem in a nonglobal zone.

Wireshark and TShark: tcpdump on steroids

tcpdump has been around since approximately the dawn of time, but a newer open source package called Wireshark (formerly known as Ethereal) has been gaining ground rapidly. Wireshark is under active development and incorporates more functionality than most commercial sniffing products. It's an incredibly powerful analysis tool and should be included in every networking expert's tool kit. It's also an invaluable learning aid.

Wireshark includes both a GUI interface (**wireshark**) and a command-line interface (**tshark**). Linux distributions make it a snap to install. UNIX administrators should check wireshark.org, which hosts the source code and a variety of precompiled binaries.

Wireshark can read and write trace files in the formats used by many other packet sniffers. Another handy feature is that you can click on any packet in a TCP conversation and ask Wireshark to reassemble (splice together) the payload data of all the packets in the stream. This feature is useful if you want to examine the data transferred during a complete TCP exchange, such as a connection used to transmit an email message across the network.

Wireshark's capture filters are functionally identical to **tcpdump**'s since Wireshark uses the same underlying **libpcap** library. Watch out, though—one important gotcha with Wireshark is the added feature of "display filters," which affect what you see rather than what's actually captured by the sniffer. The display filter syntax is more powerful than the **libpcap** syntax supported at capture time. The display filters do look somewhat similar, but they are not the same.

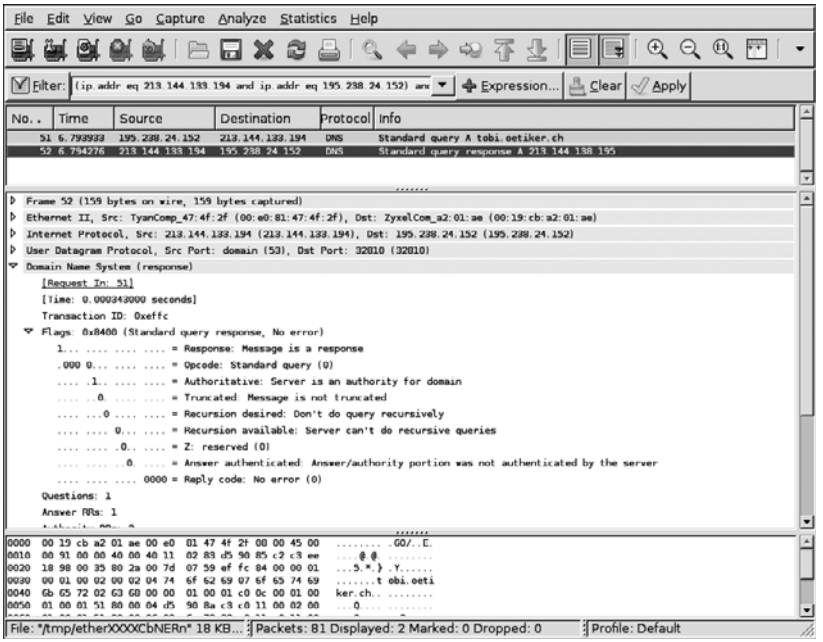
See page 274 for more information about SANs.

Wireshark has built-in dissectors for a wide variety of network protocols, including many used to implement SANs. It breaks packets into a structured tree of information in which every bit of the packet is described in plain English.

Exhibit B on the next page shows Wireshark's capture of a DNS query and response. The table near the top of the screen shows the two packets involved. The first packet is the request on its way to the DNS server, and the second packet is the answer coming back. The response packet is selected, so the middle panel shows its disassembly. The lower panel shows the packet in the form of raw bytes.

The expanded section of the tree shows the packet's DNS payload. The raw content can also be interesting to look at because it sometimes contains telltale text fragments that hint at what is going on. Scanning the text is especially handy when there is no built-in dissector for the current protocol. Wireshark's help menu provides many great examples to get you started. Experiment!

Exhibit B A pair of DNS packets in Wireshark



A note of caution regarding Wireshark: although it has lots of neat features, it has also required many security updates over the years. Run a current copy, and do not leave it running indefinitely on sensitive machines; it might be a potential route of attack.

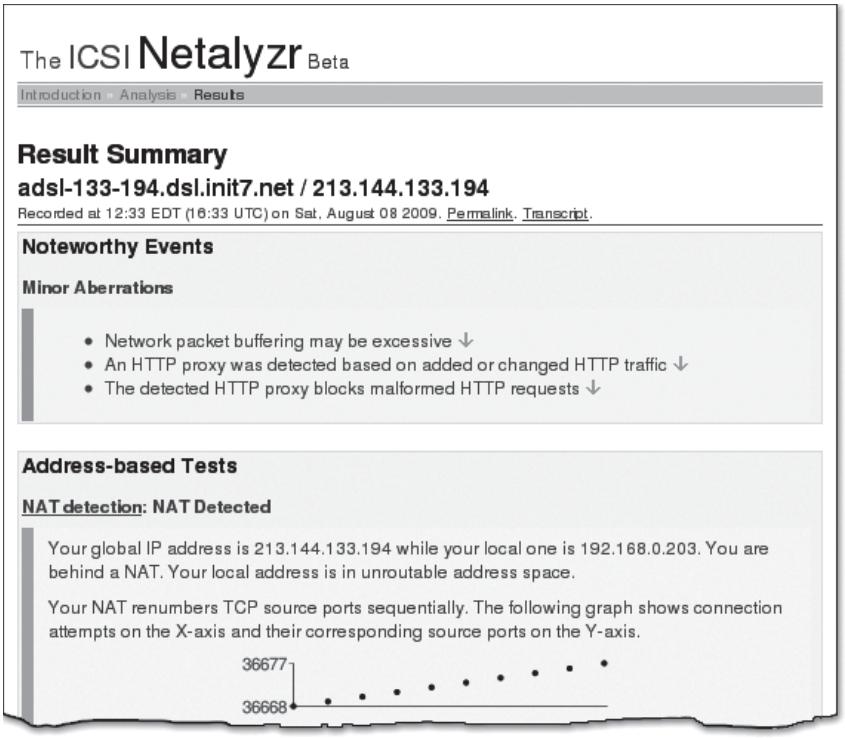
21.8 THE ICSI NETALYZR

We have looked at several tools for network debugging and for reviewing specific aspects of the network configuration. But even with your own best efforts at monitoring, it's useful to have someone else take a peek at your network from time to time. The Netalyzr is a service provided by the International Computer Science Institute at Berkeley that provides a useful "second opinion." To use it, just point your Java-enabled browser at netalyzr.icsi.berkeley.edu (note: missing 'e').

The Netalyzr tests your Internet connection in a variety of ways. It has the advantage of being able to access your network both from inside (through the Java program that runs in your browser) and from the perspective of ICSI's servers.

Exhibit C shows the Netalyzr report for a workstation on a private network that's attached to the outside world through a DSL link. The Netalyzr seems to be generally happy with the setup except for a few quibbles about the Apache web proxy that's in use. (Blocking malformed HTTP requests may actually be a useful feature, however.)

Exhibit C A Netalyzr report



The full report contains sections that report on the environment's IP connectivity, bandwidth, latency, buffering, and handling of fragmented packets, among other topics. The tests for DNS and HTTP anomalies are particularly strong.

21.9 NETWORK MANAGEMENT PROTOCOLS

Networks have grown rapidly in size and value over the last 20 years, and along with that growth has come the need for an efficient way to manage them. Commercial vendors and standards organizations have approached this challenge in many different ways. The most significant developments have been the introduction of several standard device-management protocols and a glut of high-level products that exploit those protocols.

Network management protocols standardize a way of probing a device to discover its configuration, health, and network connections. In addition, they allow some of this information to be modified so that network management can be standardized across different kinds of machinery and performed from a central location.

The most common protocol used with TCP/IP is the Simple Network Management Protocol, SNMP. Despite its name, SNMP is actually quite complex. It

defines a hierarchical namespace of management data and a way to read and write the data at each node. It also defines a way for managed servers and devices ("agents") to send event notification messages ("traps") to management stations.

The protocol itself is simple; most of SNMP's complexity lies above the protocol layer in the conventions for constructing the namespace and in the unnecessarily baroque vocabulary that surrounds SNMP like a protective shell. As long as you don't think too hard about its internal mechanics, SNMP is easy to use.

Several other standards are floating around out there. Many of them originate from the Distributed Management Task Force (DMTF), which is responsible for concepts such as WBEM (Web-Based Enterprise Management), DMI (Desktop Management Interface), and the CIM (Conceptual Interface Model). Some of these concepts, particularly DMI, have been embraced by several major vendors and may eventually become a useful complement to (or even a replacement for) SNMP. For now, however, the vast majority of networking gear management takes place over SNMP.

Since SNMP is only an abstract protocol, you need both a server program ("agent") and a client ("manager") to use it. Perhaps counterintuitively, the server side of SNMP represents the thing being managed, and the client side is the manager. Clients range from simple command-line utilities to dedicated management stations that graphically display networks and faults in eye-popping color.

Dedicated network management stations are the primary reason for the existence of management protocols. Most products let you build a topographic model of the network as well as a logical model; the two are presented together on-screen, along with a continuous indication of the status of each component.

Just as a chart can reveal the hidden meaning in a page of numbers, a network management station can summarize the state of a large network in a way that's easily accepted by a human brain. This kind of executive summary is almost impossible to get in any other way.

A major advantage of network management by protocol is that it promotes all kinds of network hardware onto a level playing field. UNIX and Linux systems are all basically similar, but routers, switches, and other low-level components are not. With SNMP, they all speak a common language and can be probed, reset, and configured from a central location. It's nice to have one consistent interface to all the network's hardware.

21.10 SNMP: THE SIMPLE NETWORK MANAGEMENT PROTOCOL

When SNMP first became widely used in the early 1990s, it started a mini gold rush. Hundreds of companies came out with SNMP management packages. Most pieces of network hardware that are intended for production use (as opposed to household use) now incorporate an SNMP agent.

Before we dive into the gritty details of SNMP, we should note that the terminology associated with it is some of the most wretched technobabble to be found in the networking arena. The standard names for SNMP concepts and objects actively lead you away from an understanding of what's going on. The people responsible for this state of affairs should have their keyboards smashed.

SNMP organization

SNMP data is arranged in a standardized hierarchy. This enforced organization allows the data space to remain both universal and extensible, at least in theory. Large portions are set aside for future expansion, and vendor-specific additions are localized to prevent conflicts. The naming hierarchy is made up of "Management Information Bases" (MIBs), structured text files that describe the data accessible through SNMP. MIBs contain descriptions of specific data variables, which are referred to with names known as object identifiers, or OIDs.

Translated into English, this means that SNMP defines a hierarchical namespace of variables whose values are tied to "interesting" parameters of the system. An OID is just a fancy way of naming a specific managed piece of information.

The SNMP hierarchy is much like a filesystem. However, a dot is used as the separator character, and each node is given a number rather than a name. By convention, nodes are also given text names for ease of reference, but this naming is really just a high-level convenience and not a feature of the hierarchy. (It is similar in principle to the mapping of hostnames to IP addresses.)

For example, the OID that refers to the uptime of the system is 1.3.6.1.2.1.1.3. This OID is also known by the human-readable (though not necessarily "human-understandable without additional documentation") name

iso.org.dod.internet.mgmt.mib-2.system.sysUpTime

The top levels of the SNMP hierarchy are political artifacts and generally do not contain useful data. In fact, useful data can currently be found only beneath the OID iso.org.dod.internet.mgmt (numerically, 1.3.6.1.2).

The basic SNMP MIB for TCP/IP (MIB-I) defines access to common management data: information about the system, its interfaces, address translation, and protocol operations (IP, ICMP, TCP, UDP, and others). A later and more complete reworking of this MIB (called MIB-II) is defined in RFC1213. Most vendors that provide an SNMP server support MIB-II. Table 21.1 on the next page presents a sampling of nodes from the MIB-II namespace.

In addition to the basic MIB, there are MIBs for various kinds of hardware interfaces and protocols, MIBs for individual vendors, and MIBs for particular hardware products.

A MIB is only a schema for naming management data. To be useful, a MIB must be backed up with agent-side code that maps between the SNMP namespace and the device's actual state. SNMP agents that run on UNIX, Linux, or Windows

Table 21.1 Selected OIDs from MIB-II

OID[a]	Type	Contents
system.sysDescr	string	System info: vendor, model, OS type, etc.
system.sysLocation	string	Physical location of the machine
system.sysContact	string	Contact info for the machine's owner
system.sysName	string	System name, usually the full DNS name
interfaces.ifNumber	int	Number of network interfaces present
interfaces.ifTable	table	Table of infobits about each interface
ip.ipForwarding	int	1 if system is a gateway; otherwise, 2
ip.ipAddrTable	table	Table of IP addressing data (masks, etc.)
ip.ipRouteTable	table	The system's routing table
icmp.icmpInRedirects	int	Number of ICMP redirects received
icmp.icmpInEchos	int	Number of pings received
tcp.tcpConnTable	table	Table of current TCP connections
udp.udpTable	table	Table of UDP sockets with servers listening

a. Relative to iso.org.dod.internet.mgmt.mib-2.

come with built-in support for MIB-II. Most can be extended to support supplemental MIBs and to interface with scripts that do the actual work of fetching and storing these MIBs' associated data.

SNMP agents are complex beasts, and they have seen their share of security issues. Instead of relying on whatever agent your vendor happens to toss over the wall to you, it may be prudent to compile and install a current copy of NET-SNMP. See *The NET-SNMP agent*, opposite page, for details.

SNMP protocol operations

There are only four basic SNMP operations: get, get-next, set, and trap.

Get and set are the basic operations for reading and writing data to the node identified by a specific OID. Get-next steps through a MIB hierarchy and can read the contents of tables as well.

A trap is an unsolicited, asynchronous notification from server (agent) to client (manager) that reports the occurrence of an interesting event or condition. Several standard traps are defined, including "I've just come up" notifications, reports of failure or recovery of a network link, and announcements of various routing and authentication problems. Many other not-so-standard traps are in common use, including some that simply watch the values of other SNMP variables and fire off a message when a specified range has been exceeded. The mechanism by which the destinations of trap messages are specified depends on the implementation of the agent.

Since SNMP messages can potentially modify configuration information, some security mechanism is needed. The simplest version of SNMP security is based on

the concept of an SNMP "community string," which is really just a horribly obfuscated way of saying "password." There's usually one community string for read-only access and another that allows writing.

Although many organizations still use the original community-string-based authentication, version 3 of the SNMP standard introduced access control methods with higher security. Although configuring this more advanced security requires a little extra work, the risk reduction is well worth the effort. If for some reason you can't use version 3 SNMP security, at least be sure you've selected a hard-to-guess community string.

RMON: remote monitoring MIB

The RMON MIB permits the collection of generic network performance data (that is, data not tied to any one particular device). Network sniffers or "probes" can be deployed around the network to gather information about utilization and performance. Once a useful amount of data has been collected, statistics and interesting information about the data can be shipped back to a central management station for analysis and presentation. Many probes have a packet capture buffer and can provide a sort of remote **tcpdump** facility.

RMON is defined in RFC1757, which became a draft standard in 1995. The MIB is broken up into nine "RMON groups." Each group contains a different set of network statistics. If you have a large network with many WAN connections, consider buying probes to reduce the SNMP traffic across your WAN links. Once you have access to statistical summaries from the RMON probes, there's usually no need to gather raw data remotely. Many switches and routers support RMON and store at least some network statistics.

21.11 THE NET-SNMP AGENT

When SNMP was first standardized, Carnegie Mellon University and MIT both produced implementations. CMU's implementation was more complete and quickly became the de facto standard. When active development at CMU died down, researchers at UC Davis took over the software. After stabilizing the code, they rehomed it at the SourceForge repository. The package is now known as NET-SNMP, and it is the authoritative free SNMP implementation for UNIX and Linux. The latest version is available from net-snmp.sourceforge.net.

NET-SNMP includes an agent, some command-line tools, a server for receiving traps, and even a library for developing SNMP-aware applications. We discuss the agent in some detail here, and on page 885 we look at the command-line tools.

As in other implementations, the agent collects information about the local host and serves it to SNMP managers across the network. The default installation includes MIBs for network interfaces, memory, disk, processes, and CPU. The agent is easily extensible since it can execute an arbitrary command and return the

command's output as an SNMP response. You can use this feature to monitor almost anything on your system with SNMP.

By default, the agent is installed as **/usr/sbin/snmpd**. It is usually started at boot time and reads its configuration information from files in the **/etc/snmp** directory. The most important of these files is **snmpd.conf**, which contains most of the configuration information and ships with a bunch of sample data collection methods enabled. Although the intention of the authors seems to have been for users to edit only the **snmpd.local.conf** file, you must edit **snmpd.conf** at least once to disable any default data collection methods that you don't plan to use.

The NET-SNMP **configure** script lets you specify a default log file and a couple of other local settings. You can use **snmpd -l** to specify an alternative log file or **-s** to direct log messages to syslog. Table 21.2 lists **snmpd**'s most important flags. We recommend that you always use the **-a** flag. For debugging, you should use the **-V**, **-d**, or **-D** flags, each of which gives progressively more information.

Table 21.2 Useful flags for NET-SNMP snmpd

Flag	Function
-l *logfile*	Logs information to *logfile*
-a	Logs the addresses of all SNMP connections
-d	Logs the contents of every SNMP packet
-V	Enables verbose logging
-D	Logs debugging information (lots of it)
-h	Displays all arguments to **snmpd**
-H	Displays all configuration file directives
-A	Appends to the log file instead of overwriting it
-s	Logs to syslog (uses the daemon facility)

It's worth mentioning that many useful SNMP-related Perl, Ruby, and Python modules are available from the respective module repositories.

21.12 NETWORK MANAGEMENT APPLICATIONS

We begin this section by exploring the simplest SNMP management tools: the commands provided with the NET-SNMP package. These commands can familiarize you with SNMP, and they're also great for one-off checks of specific OIDs. Next, we look at Cacti, a program that generates beautiful historical graphs of SNMP values, and Nagios, an event-based monitoring system. We conclude with some recommendations of what to look for when purchasing a commercial network monitoring system.

The NET-SNMP tools

Even if your system comes with its own SNMP server, you may still want to compile and install the client-side tools from the NET-SNMP package. Table 21.3 lists the most commonly used tools.

Table 21.3 Command-line tools in the NET-SNMP package

Command	Function
snmpdelta	Monitors changes in SNMP variables over time
snmpdf	Monitors disk space on a remote host via SNMP
snmpget	Gets the value of an SNMP variable from an agent
snmpgetnext	Gets the next variable in sequence
snmpset	Sets an SNMP variable on an agent
snmptable	Gets a table of SNMP variables
snmptranslate	Searches for and describes OIDs in the MIB hierarchy
snmptrap	Generates a trap alert
snmpwalk	Traverses a MIB starting at a particular OID

In addition to their value on the command line, these programs are tremendously handy in simple scripts. It is often helpful to have **snmpget** save interesting data values to a text file every few minutes. (Use **cron** to implement the scheduling; see Chapter 9, *Periodic Processes*.)

snmpwalk is another useful tool. Starting at a specified OID (or at the beginning of the MIB, by default), this command repeatedly makes "get next" calls to an agent. This behavior results in a complete list of available OIDs and their associated values. **snmpwalk** is particularly handy when you are trying to identify new OIDs to monitor from your fancy enterprise management tool.

Here's a truncated sample **snmpwalk** of the host tuva. The community string is "secret813community", and **-v1** specifies simple authentication.

```
$ snmpwalk -c secret813community -v1 tuva
SNMPv2-MIB::sysDescr.0 = STRING: Linux tuva.atrust.com 2.6.9-11.ELsmp #1
SNMPv2-MIB::sysUpTime.0 = Timeticks: (1442) 0:00:14.42
SNMPv2-MIB::sysName.0 = STRING: tuva.atrust.com
IF-MIB::ifDescr.1 = STRING: lo
IF-MIB::ifDescr.2 = STRING: eth0
IF-MIB::ifDescr.3 = STRING: eth1
IF-MIB::ifType.1 = INTEGER: softwareLoopback(24)
IF-MIB::ifType.2 = INTEGER: ethernetCsmacd(6)
IF-MIB::ifType.3 = INTEGER: ethernetCsmacd(6)
IF-MIB::ifPhysAddress.1 = STRING:
IF-MIB::ifPhysAddress.2 = STRING: 0:11:43:d9:1e:f5
IF-MIB::ifPhysAddress.3 = STRING: 0:11:43:d9:1e:f6
```

```
IF-MIB::ifInOctets.1 = Counter32: 2605613514
IF-MIB::ifInOctets.2 = Counter32: 1543105654
IF-MIB::ifInOctets.3 = Counter32: 46312345
IF-MIB::ifInUcastPkts.1 = Counter32: 389536156
IF-MIB::ifInUcastPkts.2 = Counter32: 892959265
IF-MIB::ifInUcastPkts.3 = Counter32: 7712325
…
```

In this example, we see some general information about the system, followed by statistics about the host's network interfaces: lo0, eth0, and eth1. Depending on the MIBs supported by the agent you are managing, a complete dump can run to hundreds of lines.

SNMP data collection and graphing

Network-related data is best appreciated in visual and historical context. It's important to have some way to track and graph performance metrics, but your exact choice of software for doing this is not critical.

One of the most popular early SNMP polling and graphing packages was MRTG, written by Tobi Oetiker. MRTG is written mostly in Perl, runs regularly out of **cron**, and can collect data from any SNMP source. Each time the program runs, new data is stored and new graph images are created.

Another useful tool in this area is RRDtool, also by Tobi Oetiker. It is an application tool kit for storing and graphing performance metrics. All the leading open source monitoring solutions are based on RRDtool, including our favorite, Cacti.

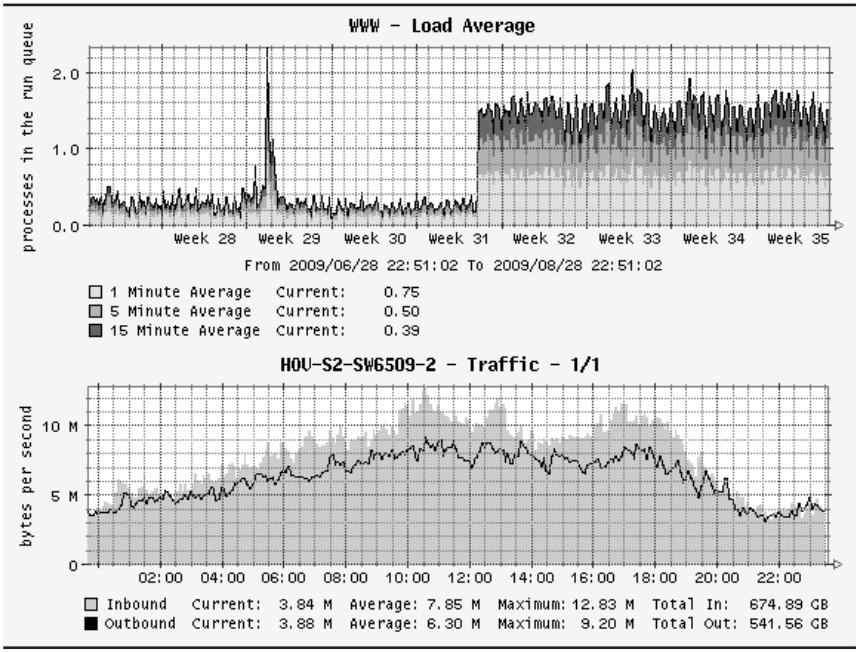
Cacti, available from cacti.net, offers several attractive features. Using RRDtool as its back end, it stores monitoring data in zero-maintenance, statically sized databases. Cacti stores only enough data to create the graphs you want. For example, Cacti could store one sample every minute for a day, one sample every hour for a week, and one sample every week for a year. This consolidation scheme lets you maintain important historical information without having to store unimportant details or consume your time with database administration.

Second, Cacti can record and graph any SNMP variable, as well as many other performance metrics. You're free to collect whatever data you want. When combined with the NET-SNMP agent, Cacti generates a historical perspective on almost any system or network resource.

Exhibit D shows some examples of the graphs created by Cacti. These graphs show the load average on a server over a period of multiple weeks along with a day's traffic on a network interface.

Cacti sports easy web-based configuration as well as all the other built-in benefits of RRDtool, such as low maintenance and beautiful graphing. See the RRDtool home page at rrdtool.org for links to the current versions of RRDtool and Cacti, as well as dozens of other monitoring tools.

Exhibit D Examples of Cacti graphs



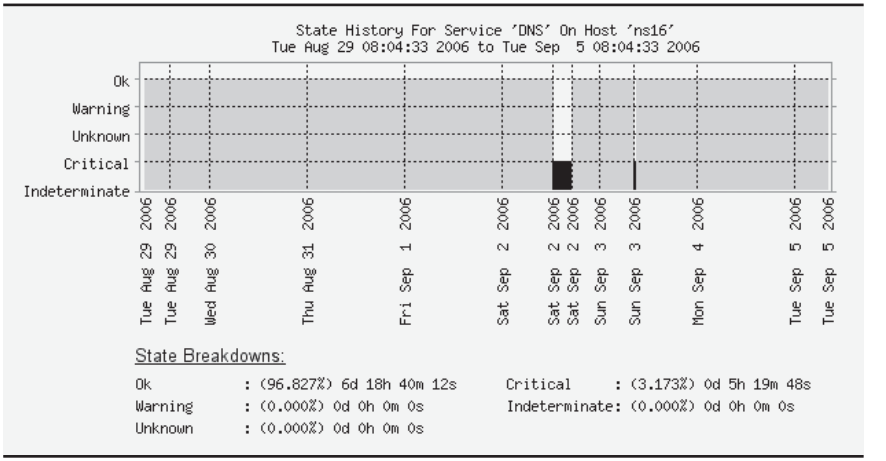
Nagios: event-based service monitoring

Nagios specializes in real-time reporting of error conditions. It includes scores of scripts for monitoring services of all shapes and sizes, along with extensive SNMP monitoring capabilities. Perhaps its greatest strength is its modular, heavily customizable configuration system that allows custom scripts to be written to monitor any conceivable metric. Although Nagios does not help you determine how much your bandwidth utilization has increased over the last month, it can page you when your web server goes down.

The Nagios distribution includes plug-ins that supervise a variety of common points of failure. You can whip up new monitors in Perl, or even in C if you are feeling ambitious. For notification methods, the distribution can send email, generate web reports, and use a dial-up modem to page you. As with the monitoring plug-ins, it's easy to roll your own.

In addition to sending real-time notifications of service outages, Nagios keeps a historical archive of this data. It provides several powerful reporting interfaces that track availability and performance trends. Many organizations use Nagios to measure compliance with service level agreements; Exhibit E on the next page shows the availability of a DNS server.

Exhibit E Server availability as shown by Nagios



Nagios works very well for networks of fewer than a thousand hosts and devices. It is easy to customize and extend, and it includes powerful features such as redundancy, remote monitoring, and escalation of notifications. If you cannot afford a commercial network management tool, you should strongly consider Nagios. You can read more at nagios.org.

The ultimate network monitoring package: still searching

As we reviewed the state of network management packages for this edition of the book, we found the software landscape bustling with activity, just as it has been for most of the last decade. However, most packages are still using RRDtool somewhere in their guts to do their logging and graphing. No high-level standard akin to **vi** or **emacs** has yet arrived on the scene.

Two well-funded companies based on the "open source plus" model (GroundWork Open Source and Zenoss) have debuted network management packages backed by serious advertising dollars and polished interfaces. In the traditional free software arena, the packages Munin (munin.projects.linpro.no) and **collectd** (collectd.org) have gained quite a following.

Munin is especially popular in the Scandinavian countries. It's built on a clever architecture in which the data collection plug-ins not only provide data but also tell the system how the data should be presented.

collectd is written in C for performance and portability. It runs even on tiny systems without hampering performance or requiring any additional dependencies. At the time of this writing, **collectd** comes with over 70 data collection plug-ins.

Commercial management platforms

Hundreds of companies sell network management software, and new competitors enter the market every week. Instead of recommending the hottest products of the moment (which may no longer exist by the time this book is printed), we identify the features you should look for in a network management system.

Data-gathering flexibility: Management tools must be able to collect data from sources other than SNMP. Many packages include ways to gather data from almost any network service. For example, some packages can make SQL database queries, check DNS records, and connect to web servers.

User interface quality: Expensive systems often offer a custom GUI or a web interface. Most well-marketed packages today tout their ability to understand XML templates for data presentation. A UI is not just more marketing hype—you need an interface that relays information clearly, simply, and comprehensibly.

Value: Some management packages come at a stiff price. HP's OpenView is both one of the most expensive and one of the most widely adopted network management systems. Many corporations find definite value in being able to say that their site is managed by a high-end commercial system. If that isn't so important to your organization, you should look at the other end of the spectrum for free tools like Cacti and Nagios.

Automated discovery: Many systems offer the ability to "discover" your network. Through a combination of broadcast pings, SNMP requests, ARP table lookups, and DNS queries, they identify all your local hosts and devices. All the discovery implementations we have seen work pretty well, but none are very accurate on a complex (or heavily firewalled) network.

Reporting features: Many products can send alert email, activate pagers, and automatically generate tickets for popular trouble-tracking systems. Make sure that the platform you choose accommodates flexible reporting; who knows what electronic devices you will be dealing with in a few years?

Configuration management: Some solutions step far beyond monitoring and alerting. They enable you to manage actual host and device configurations. For example, a CiscoWorks interface lets you change a router's configuration in addition to monitoring its state with SNMP. Because device configuration information deepens the analysis of network problems, we predict that many packages will develop along these lines in the future.

21.13 NetFlow: connection-oriented monitoring

SNMP is widely known for its ability to report the amount of network traffic flowing through an interface. But if you want to know more about the exact type of traffic and its destinations, SNMP is not much help. On a UNIX box you could run a sniffer to unearth some additional details, but this option isn't available on a dedicated router.

In response, router vendors have come up with their own solutions to this problem. The most popular of these solutions is Cisco's NetFlow protocol.

NetFlow tracks every connection with seven keys: source and destination IP address, source and destination port number, protocol (TCP, UDP, etc.), type of service (ToS), and logical interface. This metadata, combined with additional information such as the number of packets and bytes involved, can be sent to any suitable collector.

The predominant NetFlow protocol versions are v5 and v7, which are usually lumped together because they're the same except that v7 adds an additional field (source router). v7 is used on Cisco Catalyst switches. Version 9 is gaining popularity. Its template-based nature makes it very flexible.

You can have your NetFlow router send a running account of its metadata to a suitable receiver such as CAIDA's **cflowd**. On a busy network link, this configuration generates a huge amount of data, so you may need to provision substantial disk space and look into analysis tools that are up to the task.

For the latter, one possibility is Dave Plonka's FlowScan package. It has unfortunately not been updated in some time, but it still works well. You can find it at net.doit.wisc.edu/~plonka/FlowScan.

Monitoring NetFlow data with nfdump and NfSen

Another pair of useful tools for collecting and analyzing NetFlow data are Peter Haag's **nfdump** (nfdump.sourceforge.net) and NfSen (nfsen.sourceforge.net). The collector (**nfcapd**) stores NetFlow data on disk for later processing by **nfdump**.

nfcapd and **nfdump** handle NetFlow protocol versions v5/v7 and v9. For IPv6 support, you'll have to use v9; versions 5 and 7 do not support it.

nfdump works a bit like **tcpdump** (see page 875). It has a similar filter syntax that has been adapted for NetFlow data. Flexible output formats let you customize the display of records. Built-in summarizers show you the top N talkers[7] on your network and other useful information.

The following (slightly condensed) **nfdump** output shows which IP addresses and networks exchange the most traffic, which ports are currently the most active, and

7. A "talker" is the NetFlow term for a device that creates network traffic.

more. The **-s ip/flows** option asks for information about any source or destination IP address, sorted by flows. **-n 10** limits the display to the top 10 items.

```
linux$ nfdump -M /data/nfsen/profiles-data/live/upstream
    -r 2009/07/28/12/nfcapd.200907281205 -n 10 -s ip/flows
Top 10     IP Addr ordered by flows:
Date first seen    Durat'n  IP Addr          Flows  Pkts    Bytes   pps  bps     bpp
2009-07-28 12:02  467.596  192.168.96.92    27873  67420   3.8 M   144  67347   58
2009-07-28 12:02  462.700  192.168.96.107   18928  43878   4.7 M   94   85522   112
2009-07-28 12:02  464.443  192.168.96.198   17321  45454   3.5 M   97   63884   81
2009-07-28 12:02  454.299  172.16.152.40    11554  29093   1.3 M   64   23996   46
2009-07-28 12:02  362.586  192.168.97.203   6839   11104   1.2 M   30   28883   117
2009-07-28 12:02  393.600  172.16.220.139   4802   12883   618384  32   12568   48
2009-07-28 12:02  452.353  192.168.96.43    4477   5144    554709  11   9810    107
2009-07-28 12:02  456.306  192.168.96.88    3416   6642    697776  14   12233   105
2009-07-28 12:02  459.732  192.168.96.108   2544   25555   3.2 M   55   58478   131
2009-07-28 12:02  466.782  192.168.96.197   2143   24103   5.3 M   51   94988   229

Summary: total flows: 98290, total bytes: 311.6 M, total packets: 759205, avg
    bps: 5.3 M, avg pps: 1623, avg bpp: 430
Time window: 2009-07-28 12:02:12 - 2009-07-28 12:09:59
Total flows processed: 98290, skipped: 0, Bytes read: 5111164
Sys: 0.310s flows/second: 317064.5   Wall: 0.327s flows/second: 300366.1
```

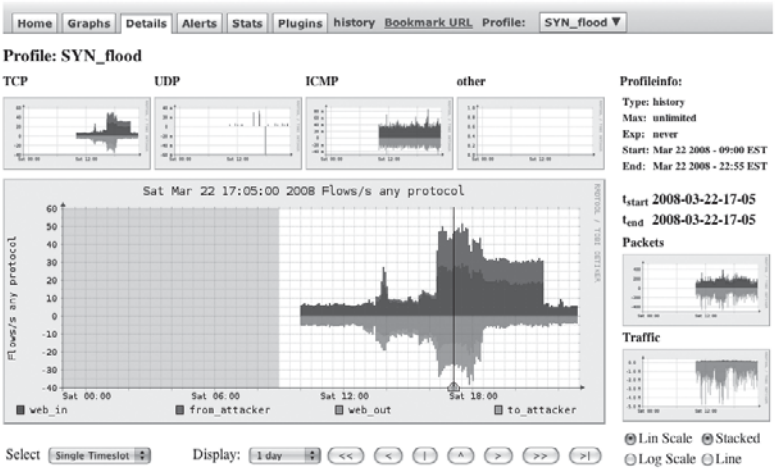
Since the NetFlow data is stored on disk, you can analyze it repeatedly with different sets of filters. Another nice feature is **nfdump**'s ability to match incoming and outgoing flows into a single bidirectional flow.

NfSen is a web front end for NetFlow data that sits on top of **nfdump** and therefore combines graphing capabilities with all the features of **nfdump**. It displays the data in three different categories: flows, packets, and bytes. NfSen does more than just create static graphs, though—it lets you navigate through data, point to interesting peaks in the graphs, and drill down to the individual flows. You can also apply arbitrary **nfdump** filters to refine the display. The combination of easy GUI browsing with the underlying power of **nfdump** makes NfSen a powerful tool.

NfSen lets you save your filter and display settings together as a profile so that you can easily return to a specific type of analysis in the future. For example, you might define profiles that monitor traffic for your DMZ, your web server, or a client's network.

Profiles also make NfSen a valuable tool for security incident response teams because they make it easy to track specific types of incidents or network traffic. For example, Exhibit F on the next page shows a display that's customized for investigating "SYN flood" denial of service attacks.

Exhibit F A "SYN flood" profile for NfSen



A security investigation usually happens hours or days after the incident that triggered it, but if you save NetFlow data as a matter of course, you can easily create an NfSen profile that looks back to an earlier time period. This retrospective view lets you identify the IP addresses involved in an attack and track down other hosts that may have been affected. You can also set up NfSen to watch your flows outside of office hours and to trigger alarms when certain conditions are met.

Setting up NetFlow on a Cisco router

To get started with NetFlow, you must first configure your network device to send NetFlow data to **nfcapd**. This section outlines the configuration of NetFlow on a Cisco router.

Export of NetFlow data is enabled per interface:

```
ios# interface fastethernet 0/0
ios# ip route-cache flow
```

To tell the router where to send the NetFlow data, enter the following command:

```
ios# ip flow-export nfcapd-hostname listen-port
```

The options below break up long-lived flows into 5-minute segments. You can choose any segment length between 1 and 60 minutes, but it should be equal to or less than **nfdump**'s file rotation period, which is 5 minutes by default.

```
ios# ip flow-export version 5
ios# ip flow-cache timeout active 5
```

On the Catalyst 6500/7600, you must enable NDE (NetFlow Data Export) in addition to normal NetFlow export.

Here's how:

```
ios# mls flow ip interface-full
ios# mls flow ipv6 interface-full
ios# mls nde sender version 5
```

On a busy router, consider aggressively timing out small flows:

```
ios# mls aging fast time 4 threshold 2
ios# mls aging normal 32
ios# mls aging long 900
```

You still need the traditional NetFlow configuration, including **ip flow ingress** or **ip route-cache flow** on every interface, so that you see "software switched" flows such as those that go to the router itself.

For NetFlow v9, the configuration may be even longer. Depending on your IOS version, you can also define your own template. With the introduction of Flexible NetFlow (FNF), the NetFlow environment has become even more complex.

21.14 RECOMMENDED READING

Wikipedia includes a nice (though somewhat compressed) overview of SNMP with pointers to RFCs. It's a good starting point.

MAURO, DOUGLAS R., AND KEVIN J. SCHMIDT. *Essential SNMP (2nd Edition)*. Sebastopol, CA: O'Reilly Media, 2005.

SIMPLEWEB. *SNMP and Internet Management Site*. simpleweb.org.

You may find the following RFCs to be useful as well. We replaced the actual titles of the RFCs with a description of the RFC contents because some of the actual titles are an unhelpful jumble of buzzwords and SNMP jargon.

- RFC1155 – Characteristics of the SNMP data space (data types, etc.)
- RFC1156 – MIB-I definitions (description of the actual OIDs)
- RFC1157 – Simple Network Management Protocol
- RFC1213 – MIB-II definitions (OIDs)
- RFC3414 – User-based Security Model for SNMPv3
- RFC3415 – View-based Access Control Model for SNMPv3
- RFC3512 – Configuring devices with SNMP (best general overview)
- RFC3584 – Practical coexistence between different SNMP versions
- RFC3954 - Cisco Systems NetFlow Services Export Version 9

22 Security

Despite Hollywood's best efforts, the maintenance of a secure computing environment remains unglamorous and largely unappreciated. It is a system administration discipline born of necessity; if UNIX and Linux systems are going to house sensitive data and control critical processes, we *must* protect them.

Such protection requires resources, both in terms of sysadmin time and in the hard currency of security-related equipment. Unfortunately, many organizations don't make the appropriate investments in this area until an incident has already occurred.

In November 1988, we experienced our first real taste of the security threat posed by a world-wide network as the Robert Morris, Jr., Internet worm was unleashed onto the Internet (see the Wikipedia article on "Morris worm"). Before that event, the Internet lived in an age of innocence. Security was a topic that administrators thought about mostly in the "what if" sense. A big security incident usually consisted of something like a user gaining administrative access to read another user's mail, often just to prove that he could.

The Morris worm wasted thousands of administrator hours but greatly increased security awareness on the Internet. Once again, we were painfully reminded that good fences make good neighbors. A number of excellent tools for use by system

administrators (as well as a formal organization for handling incidents of this nature) came into being as a result.

Today, security breaches are commonplace. According to the 2008 CSI/FBI *Computer Crime and Security Survey,*[1] responding organizations reported an average annual loss ascribable to security breaches of $234,000. Most large organizations report having at least one significant security breach each year.

Addressing this problem isn't as easy as you might think. Security is not something that you can buy in a box or as a service from a third party. Commercial products and services can be part of a solution for your site, but they are not a panacea. Achieving an acceptable level of security requires an enormous amount of patience, vigilance, knowledge, and persistence—not just from you and other administrators, but from your entire user and management communities.

As the system administrator, you must personally ensure that your systems are secure, that they are vigilantly monitored, and that you and your users are properly educated. You should familiarize yourself with current security technology, actively monitor security mailing lists, and hire professional security experts to help with problems that exceed your knowledge.

22.1 Is UNIX secure?

Of course not. Neither UNIX nor Linux is secure, nor is any other operating system that communicates on a network. If you must have absolute, total, unbreachable security, then you need a measurable air gap[2] between your computer and any other device. Some people argue that you also need to enclose your computer in a special room that blocks electromagnetic radiation (Wikipedia: "Faraday cage"). How fun is that?

You can work to make your system somewhat more resistant to attack. Even so, several fundamental flaws in the UNIX model ensure that you will never reach security nirvana:

- UNIX is optimized for convenience and doesn't make security easy or natural. The system's overall philosophy stresses easy manipulation of data in a networked, multiuser environment.
- The software that runs on UNIX systems is developed by a large community of programmers. They range in experience level, attention to detail, and knowledge of the system and its interdependencies. As a result, even the most well-intended new features can introduce large security holes.

1. This survey is conducted yearly and can be found at gocsi.com.
2. Of course, wireless networking technology introduces a whole new set of problems. Air gap in this context means "no networking whatsoever."

- Most administrative functions are implemented outside the kernel, where they can be inspected and tampered with. Hackers have broad access to the system.

On the other hand, since some systems' source code (e.g., Linux, OpenSolaris) is available to everyone, thousands of people can (and do) scrutinize each line of code for possible security threats. This arrangement is widely believed to result in better security than that of closed operating systems, in which a limited number of people have the opportunity to examine the code for holes.

Many sites are a release or two behind, either because localization is too troublesome or because they do not subscribe to a software maintenance service. In any case, when security holes are patched, the window of opportunity for hackers often does not disappear overnight.

It might seem that security should gradually improve over time as security problems are discovered and corrected, but unfortunately this does not seem to be the case. System software is growing ever more complicated, hackers are becoming better and better organized, and computers are connecting more and more intimately on the Internet. Security is an ongoing battle that can never really be won.

Remember, too, that

$$\text{Security} = \frac{1}{(1.072)(\text{Convenience})}$$

The more secure your system, the more constrained you and your users will be. Implement the security measures suggested in this chapter only after carefully considering the implications for your users.

22.2 How security is compromised

This chapter discusses some common security problems and their standard countermeasures. But before we leap into the details, we should take a more general look at how real-world security problems tend to occur. Most security lapses fit into the following taxonomy.

Social engineering

The human users (and administrators) of a computer system are the weakest links in the chain of security. Even in today's world of heightened security awareness, unsuspecting users with good intentions are easily convinced to give away sensitive information. No amount of technology can protect against the user element—you must ensure that your user community has a high awareness of security threats so that they can be part of the defense.

This problem manifests itself in many forms. Attackers cold-call their victims and pose as legitimately confused users in an attempt to get help accessing the system. Administrators unintentionally post sensitive information on public forums when

troubleshooting problems. Physical compromises occur when seemingly legitimate maintenance personnel rewire the phone closet.

The term "phishing" describes attempts to collect information from users through deceptive email, instant messages, or even SMS messages. Phishing can be especially hard to defend against because the communications often include victim-specific information that lends them the appearance of authenticity.

Social engineering continues to be a powerful hacking technique and is one of the most difficult threats to neutralize. Your site security policy should include training for new employees. Regular organization-wide communications are an effective way to provide information about telephone dos and don'ts, physical security, email phishing, and password selection.

To gauge your organization's resistance to social engineering, you might find it informative to attempt some social engineering attacks of your own. Be sure you have explicit permission to do this from your own managers, however. Such exploits look very suspicious if they are performed without a clear mandate. They're also a form of internal spying, so they have the potential to generate resentment if they're not handled in an aboveboard manner.

Many organizations find it useful to communicate to users that administrators will never request their passwords, whether by email, instant message, or telephone. Tell users to report any such password requests to the IT department immediately.

Software vulnerabilities

Over the years, countless security-sapping bugs have been discovered in computer software (including software from third parties, both commercial and free). By exploiting subtle programming errors or context dependencies, hackers have been able to manipulate systems into doing whatever they want.

Buffer overflows are a common programming error and one with complex implications. Developers often allocate a predetermined amount of temporary memory space, called a buffer, to store a particular piece of information. If the code isn't careful about checking the size of the data against the size of the container that's supposed to hold it, the memory adjacent to the allocated space is at risk of being overwritten. Crafty hackers can input carefully composed data that crashes the program or, in the worst case, executes arbitrary code.

Fortunately, the sheer number of buffer overflow exploits in recent years has raised the programming community's consciousness about this issue. Although buffer overflow problems are still occurring, they are often quickly discovered and corrected, especially in open source applications. Newer programming systems such as Java and .NET include mechanisms that automatically check data sizes and prevent buffer overflows. Sometimes.

Buffer overflows are a subcategory of a larger class of software security bugs known as input validation vulnerabilities. Nearly all programs accept some type of input from users (e.g., command-line arguments or HTML forms). If the code processes such data without rigorously checking it for appropriate format and content, bad things can happen. Consider the following simple example:

```
#!/usr/bin/perl
# Example user input validation error

open(HTMLFILE, "/var/www/html/$ARGV[0]") or die "trying\n";
while(<HTMLFILE>) { print; }
close HTMLFILE;
```

The intent of this code is probably to print the contents of some HTML file under **/var/www/html**, which is the default document root for the Apache web server on Red Hat servers. The code accepts a filename from the user and includes it as part of the argument to `open`. But if a malicious user entered **../../../etc/passwd** as the argument, the contents of **/etc/passwd** would be echoed!

What can you as an administrator do to prevent this type of attack? Very little, at least until a bug has been identified and addressed in a patch. Keeping up with patches and security bulletins is an important part of most administrators' jobs. Most Linux distributions include automated patching utilities, such as **yum** on Red Hat and **apt-get** on Ubuntu. OpenSolaris also has automated (and failsafe) updates implemented through **pkg image-update**. Take advantage of these utilities to keep your site safe from software vulnerabilities.

Configuration errors

Many pieces of software can be configured securely or not-so-securely. Unfortunately, because software is developed to be useful instead of annoying, not-so-securely is often the default. Hackers frequently gain access by exploiting software features that would be considered helpful and convenient in less treacherous circumstances: accounts without passwords, disks shared with the world, and unprotected databases, to name a few.

A typical example of a host configuration vulnerability is the standard practice of allowing Linux systems to boot without requiring a boot loader password. GRUB can be configured at install time to require a password, but administrators almost always decline the option. This omission leaves the system open to physical attack. However, it's also a perfect example of the need to balance security against usability. Requiring a password means that if the system were unintentionally rebooted (e.g., after a power outage), an administrator would have to be physically present to get the machine running again.

One of the most important steps in securing a system is simply making sure that you haven't inadvertently put out a welcome mat for hackers. Problems in this category are the easiest to find and fix, although there are potentially a lot of them and it's not always obvious what to check for. The port and vulnerability scanning

tools covered later in this chapter can help a motivated administrator identify problems before they're exploited.

22.3 SECURITY TIPS AND PHILOSOPHY

This chapter discusses a wide variety of security concerns. Ideally, you should address all of them within your environment. Most administrators should probably digest the contents of this entire chapter more than once.

Most systems do not come secured out of the box. In addition, customizations made both during and after installation change the security profile for new systems. Administrators should take steps to harden new systems, integrate them into the local environment, and plan for their long-term security maintenance.

When the auditors come knocking, it's useful to be able to prove that you have followed a standard methodology, especially if that methodology conforms to external recommendations and best practices for your industry.

We use a localization checklist to secure new systems. A system administrator applies the standard hardening steps to the system, and a security administrator then confirms that the steps were followed correctly and keeps a log of newly secured systems.

Patches

Keeping the system updated with the latest patches is an administrator's highest-value security chore. Most systems are configured to point at the vendor's repository, which makes applying patches as simple as running a few commands. Larger environments can use a local repository that mirrors that of the vendor.

A reasonable approach to patching should include the following elements:

- A regular schedule for installing routine patches that is diligently followed. Consider the impact on users when designing this schedule. Monthly updates are usually sufficient; regularity is more important than immediacy. It is not acceptable to fix high-profile zero-day vulnerabilities but neglect other updates.
- A change plan that documents the impact of each set of patches, outlines appropriate postinstallation testing steps, and describes how to back out the changes in the event of problems. Communicate this change plan to all relevant parties.
- An understanding of what patches are relevant to the environment. Administrators should subscribe to vendor-specific security mailing lists and blogs, as well as to generalized security discussion forums such as Bugtraq. An accurate inventory of applications and operating systems used in your environment helps ensure complete coverage.

Unnecessary services

Most systems come with several services configured to run by default. Be sure to disable (and possibly remove) any that are unnecessary, especially if they are network daemons. One way to see which services are running is to use the **netstat** command. Here's partial output from a Solaris system:

```
solaris$ netstat -an | grep LISTEN
      *.111                *.*       0      0  49152    0 LISTEN
      *.32771              *.*       0      0  49152    0 LISTEN
      *.32772              *.*       0      0  49152    0 LISTEN
      *.22                 *.*       0      0  49152    0 LISTEN
      *.4045               *.*       0      0  49152    0 LISTEN
```

A variety of techniques can identify the service that's using an unknown port. On most systems, **lsof** or **fuser** may be of help. Under Linux, either command can identify the PID of the process that's using a given port:

```
ubuntu$ sudo fuser 22/tcp
22/tcp:               2454  8387

ubuntu$ sudo lsof -i:22
COMMAND   PID  USER   FD   TYPE   DEVICE   SIZE  NODE  NAME
sshd     2454  root   3u   IPv4     5730         TCP   *:ssh(LISTEN)
sshd     2454  root   4u   IPv6     5732         TCP   *:ssh(LISTEN)
```

Once you have the PIDs, you can then use **ps** to identify specific processes. If the service is unneeded, stop it and make sure that it won't be restarted at boot time.

Unfortunately, the availability of **lsof** and **fuser** varies by system, and implementations differ widely. Many versions of both tools lack support for network sockets.

If **lsof** and **fuser** aren't available (or aren't useful), you can either look up "well known" service ports in the **/etc/services** file or run **netstat** without the **-n** option to let it do this lookup for you.

The security risks inherent in some network protocols render them unsafe in almost all circumstances. FTP, Telnet, and the BSD "r" programs (**rcp**, **rlogin**, and **rsh**) use insecure authentication and data transfer methods. They should be disabled on all systems in favor of more secure alternatives such as SSH.

Remote event logging

See Chapter 11 for more information about syslog.

The syslog facility forwards log information to files, lists of users, or other hosts on your network. Consider setting up a secure host to act as a central logging machine that parses forwarded events and takes appropriate action. A single centralized log aggregator can capture logs from a variety of devices and alert administrators whenever meaningful events occur. Remote logging also prevents hackers from covering their tracks by rewriting or erasing log files on systems that have been compromised.

Most systems come configured to use syslog by default, but you will need to customize the configuration to set up remote logging.

Backups

See Chapter 10 for more information about backups.

Regular system backups are an essential part of any site security plan. They fall into the "availability" bucket of the CIA triad discussed on page 944. Make sure that all partitions are regularly dumped and that you store some backups off-site. If a significant security incident occurs, you'll then have an uncontaminated checkpoint from which to restore.

Backups can also be a security hazard. A stolen collection of tapes can circumvent the rest of the system's security. When storing tapes off-site, use a fireproof safe to deter theft and consider using encryption. If you are thinking about using a contract storage facility, ask for a physical tour.

Viruses and worms

UNIX and Linux have been mostly immune from viruses. Only a handful exist (most of which are academic in nature), and none have done the costly damage that has become commonplace in the Windows world. Nonetheless, this fact hasn't stopped certain antivirus vendors from predicting the demise of the platform from malware—unless you purchase their antivirus product at a special introductory price, of course.

The exact reason for the lack of malicious software is unclear. Some claim that UNIX simply has less market share than its desktop competitors and is therefore not an interesting target for virus authors. Others insist that UNIX's access-controlled environment limits the damage from a self-propagating worm or virus.

The latter argument has some validity. Because UNIX restricts write access to system executables at the filesystem level, unprivileged user accounts cannot infect the rest of the environment. Unless the virus code is being run by root, the scope of infection is significantly limited. The moral, then, is not to use the root account for day-to-day activities.

See Chapter 20 for more information about email content scanning.

Perhaps counterintuitively, one valid reason to run antivirus software on UNIX servers is to protect your site's Windows systems from Windows-specific viruses. A mail server can scan incoming email attachments for viruses, and a file server can scan shared files for infection. However, this solution should supplement desktop antivirus protection rather than replace it.

ClamAV by Tomasz Kojm is a popular, free antivirus product for UNIX and Linux. This widely used GPL tool is a complete antivirus toolkit with signatures for thousands of viruses. You can download the latest version from clamav.net.

Trojan horses

Trojan horses are programs that aren't what they seem to be. An example of a Trojan horse is a program called **turkey** that was distributed on Usenet a long

time ago. The program said it would draw a picture of a turkey on your terminal screen, but it actually deleted files from your home directory.

Trojan fragments appear in major software packages now and then. **sendmail**, **tcpdump**, OpenSSH, and InterBase have all issued advisories regarding malicious software in their products. These Trojans typically embed malicious code that allows attackers to access the victim's systems at will. Fortunately, most vendors fix the software and issue an advisory in a week or two. Be sure to watch the security mailing lists for any network software packages you run on your hosts.

Even given the number of security-related escapades the UNIX community has seen over the last few years, it is remarkable how few Trojan horse incidents have occurred. Credit for this state of affairs is due largely to the speed of Internet communication. Obvious security problems tend to be discovered quickly and widely discussed. Malicious packages don't stay available for very long on well-known Internet servers.

You can be certain that any software that has been discovered to be malicious will cause a big stink on the Internet. Google the name of a software package before installing it and make sure the first page of results doesn't look incriminating.

Rootkits

The craftiest hackers try to cover their tracks and avoid detection. Often, they hope to continue using your system to distribute software illegally, probe other networks, or launch attacks against other systems. They often use "rootkits" to help them remain undetected. Sony's Trojan horse employed rootkit-like capabilities to hide itself from the user.

Rootkits are programs and patches that hide important system information such as process, disk, or network activity. They come in many flavors and vary in sophistication from simple application replacements (such as hacked versions of **ls** and **ps**) to kernel modules that are nearly impossible to detect.

Host-based intrusion detection software such as OSSEC is an effective way to monitor systems for the presence of rootkits. There are also rootkit finder scripts (such as **chkrootkit**, chkrootkit.org) that scan the system for known rootkits.

Although programs are available to help administrators remove rootkits from a compromised system, the time it takes to perform a thorough cleaning would probably be better spent saving data, reformatting the disk, and starting from scratch. The most advanced rootkits are aware of common removal programs and try to subvert them.

Packet filtering

If you're connecting a system to a network that has Internet access, you *must* install a packet-filtering router or firewall between the system and the outside world. As an alternative, some systems let you implement packet filtering with software on the system itself, an option we discuss starting on page 935. Whatever

the implementation, the packet filter should pass only traffic for services that you specifically want to provide or use from that system.

Passwords

We're simple people with simple rules. Here's one: every account must have a password, and it needs to be something that can't easily be guessed. It's never a good idea to send plaintext reusable passwords across the Internet. If you allow remote logins to your system, you must use SSH or some other secure remote access system (discussed starting on page 926).

Vigilance

To ensure the security of your system, you must monitor its health, network connections, process table, and overall status regularly (usually, daily). Perform regular self-assessments, using the power tools discussed later in this chapter. Security problems tend to start small and grow quickly, so the earlier you identify an anomaly, the better off you'll be.

General philosophy

Effective system security has its roots in common sense. Some rules of thumb:

- Don't put files on your system that are likely to be interesting to hackers or to nosy employees. Trade secrets, personnel files, payroll data, election results, etc., must be handled carefully if they're on-line. Securing such information cryptographically provides a far higher degree of security than simply trying to prevent unauthorized users from accessing the files that contain the juicy tidbits.
- Your site's security policy should specify how sensitive information is handled. See Chapter 32, *Management, Policy, and Politics*, and the security standards section in this chapter (page 945) for some suggestions.
- Don't provide places for hackers to build nests in your environment. Hackers often break into one system and then use it as a base of operations to get into other systems. Sometimes hackers may use your network to cover their tracks while they attack their real target. Publicly exposed services with vulnerabilities, world-writable anonymous FTP directories, shared accounts, and neglected systems all encourage nesting activity.
- Set traps to help detect intrusions and attempted intrusions. Tools such as OSSEC, Bro, Snort, and John the Ripper (described starting on page 916) keep you abreast of potential problems.
- Religiously monitor the reports generated by these security tools. A minor problem you ignore in one report may grow into a catastrophe by the time the next report is sent.

- Teach yourself about system security. Traditional know-how, user education, and common sense are the most important parts of a site security plan. Bring in outside experts to help fill in gaps, but only under your close supervision and approval.
- Prowl around looking for unusual activity. Investigate anything that seems unusual, such as odd log messages or changes in the activity of an account (more activity, activity at strange hours, or perhaps activity while the owner is on vacation).

22.4 PASSWORDS AND USER ACCOUNTS

*See page 176 for more information about the **passwd** file.*

Poor password management is a common security weakness. By default, the contents of the **/etc/passwd** and **/etc/shadow** files determine who can log in, so these files are the system's first line of defense against intruders. They must be scrupulously maintained and free of errors, security hazards, and historical baggage.

UNIX allows users to choose their own passwords, and although this is a great convenience, it leads to many security problems. When you give users their logins, you should also instruct them on how to choose a good password. Passwords should be at least eight characters long and should include numbers, punctuation, and changes in case. Nonsense words, combinations of simple words, or the first letters of words in a memorable phrase make the best passwords. (Of course, "memorable" is good but "traditional" is hacker bait; make up your own phrase.) The comments in the section *Choosing a root password* on page 111 are equally applicable to user passwords.

It is important to continually verify (preferably daily) that every login has a password. Entries in the **/etc/shadow** file that describe pseudo-users such as "daemon" who own files but never log in should have a star or an exclamation point in their encrypted password field. These do not match any password and thus prevent use of the account.

At sites that use a centralized authentication scheme such as LDAP or Active Directory, the same logic applies. Enforce password complexity requirements, and lock out accounts after a few failed login attempts.

Password aging

Most systems that have shadow passwords also allow you to compel users to change their passwords periodically, a facility known as password aging. This feature may seem appealing at first glance, but it has several problems. Users often resent having to change their passwords, and since they don't want to forget the new password, they choose something simple that is easy to type and remember. Many users switch between two passwords each time they are forced to change, or increment a digit in the password, defeating the purpose of password aging. PAM modules (see page 908) can help enforce strong passwords to avoid this pitfall.

On Linux systems, the **chage** program controls password aging. Using **chage**, administrators can enforce minimum and maximum times between password changes, password expiration dates, the number of days to warn users before their passwords expire, the number of days of inactivity that are permissible before accounts are automatically locked, and more. The following command sets the minimum number of days between password changes to 2, the maximum number to 90, the expiration date to July 31, 2010, and warns the user for 14 days that the expiration date is approaching:

```
$ sudo chage -m 2 -M 90 -E 2010-07-31 -W 14 ben
```

For more information about user account settings, see Chapter 7.

Other systems implement password aging differently, usually with less granularity. Under Solaris, you set password aging preferences in **/etc/default/password**. Password aging on HP-UX systems is controlled through the **smc** console, and in AIX it's configured in the file **/etc/security/user**.

Group logins and shared logins

Any login that is used by more than one person is bad news. Group logins (e.g., "guest" or "demo") are sure terrain for hackers to homestead and are prohibited in many contexts by federal regulations such as HIPAA. Don't allow them at your site. However, technical controls can't prevent users from sharing passwords, so education is the best enforcement tactic.

User shells

In theory, you can set the shell for a user account to be just about any program, including a custom script. In practice, the use of shells other than standards such as **bash** and **tcsh** is a dangerous practice, and the risk is even greater for passwordless logins that have a script as their shell. If you find yourself tempted to create such a login, you might consider a passphrase-less SSH key pair instead.

Rootly entries

The only distinguishing feature of the root login is its UID of zero. Since there can be more than one entry in the **/etc/passwd** file that uses this UID, there can be more than one way to log in as root.

A common way for a hacker to install a back door after having obtained a root shell is to edit new root logins into **/etc/passwd**. Programs such as **who** and **w** refer to the name stored in **utmp** rather than the UID that owns the login shell, so they cannot expose hackers that appear to be innocent users but are really logged in as UID 0.

Don't allow root to log in remotely, even through the standard root account. Under OpenSSH, you can set the `PermitRootLogin` configuration option to `No` in the **/etc/ssh/sshd_config** file to enforce this restriction.

On Solaris, you can put `CONSOLE=/dev/console` in **/etc/default/login** to prohibit root logins from locations beside the console.

Because of **sudo** (see page 113), it's rare that you'll ever need to log in as root, even on the system console.

22.5 PAM: COOKING SPRAY OR AUTHENTICATION WONDER?

PAM stands for "pluggable authentication modules." The PAM system relieves programmers of the chore of implementing authentication systems and gives sysadmins flexible, modular control over the system's authentication methods. Both the concept and the term come from Sun Microsystems (now part of Oracle) and from a 1996 paper by Samar and Lai of SunSoft.

In the distant past, commands like **login** included hardwired authentication code that prompted the user for a password, tested the password against the encrypted version obtained from **/etc/shadow** (**/etc/passwd** at that time, really), and rendered a judgment as to whether the two passwords matched. Of course, other commands (e.g., **passwd**) contained similar code. It was impossible to change authentication methods without source code, and administrators had little or no control over details such as whether the system should accept "password" as a valid password. PAM changed all of that.

PAM puts the system's authentication routines into a shared library that **login** and other programs can call. By separating authentication functions into a discrete subsystem, PAM makes it easy to integrate new advances in authentication and encryption into the computing environment. For instance, multifactor authentication can be supported without changes to the source code of **login** and **passwd**.

For the sysadmin, setting the right level of security for authentication has become a simple configuration task. Programmers win, too: they no longer have to write tedious authentication code, and more importantly, their authentication systems are implemented correctly on the first try. PAM can authenticate all sorts of activities: user logins, other forms of system access, use of protected web sites—even the configuration of applications.

System support for PAM

All of our example systems support PAM. Configuration information goes in the **/etc/pam.d** directory (Linux) or in the **/etc/pam.conf** file (Solaris, HP-UX, and AIX). The formats of the configuration files are basically the same, but the UNIX systems put everything in one file and the Linux systems have a file for each service or command that uses PAM.

PAM support is nearly universal at this point, but if you're using some other variant of UNIX and want to check whether your system uses PAM, you can run **ldd /bin/login** to see if that binary links to PAM's shared library, **libpam**.

PAM configuration

PAM configuration files are a series of one-liners, each of which names a particular PAM module to be used on the system.

The general format is

[*service*] *module-type control-flag module-path* [*arguments*]

Fields are separated by whitespace.

Linux systems don't use a *service* field, or more accurately, they put each service in its own configuration file and let the filename assume the role of the UNIX *service* parameter. The *service* can name an authentication context to which the configuration line applies (e.g., login for vanilla user logins) or can contain the keyword other to set system defaults.

Here's an illustrative snippet from a Solaris system; all *module-path* fields are relative to the **/usr/lib/security** directory.

```
# login service

login    auth  requisite      pam_authtok_get.so.1
login    auth  required       pam_dhkeys.so.1
login    auth  required       pam_unix_cred.so.1
login    auth  required       pam_unix_auth.so.1
login    auth  required       pam_dial_auth.so.1
...
```

Individual PAM modules have finer granularity than just "authenticate the user," so there may be several lines in a PAM configuration file for any given service and module type. A series of lines for a given service and module type form a "stack."

The order in which modules appear in the PAM configuration file is important. For example, the module that prompts the user for a password must come before the module that checks that password for validity. One module can pass its output to the next by setting either environment variables or PAM variables.

The *module-type* parameter—auth, account, session, or password—determines what the module is expected to do. auth modules identify the user and grant group memberships. Modules that do account chores enforce restrictions such as limiting logins to particular times of day, limiting the number of simultaneous users, or limiting the ports on which logins can occur. (For example, you would use an account-type module to restrict root logins to the console.) session chores include tasks that are done before or after a user is granted access; for example, mounting the user's home directory. Finally, password modules change a user's password or passphrase.

The *control-flag* specifies how the modules in the stack should interact to produce an ultimate result for the stack. Table 22.1 on page 910 shows the common values.

If PAM could simply return a failure code as soon as the first individual module in a stack failed, the *control-flags* system would be simpler. Unfortunately, the system is designed so that most modules get a chance to run regardless of their sibling modules' success or failure, and this fact causes some subtleties in the flow of control. (The intent is to prevent an attacker from learning which module in the PAM stack caused the failure.)

Table 22.1 PAM control flags

Flag	Stop on failure?	Stop on success?	Comments
binding[a]	No	Yes	Like sufficient, but can't fail without failing the stack
include[a]	–	–	Includes another config file at this point in the stack
optional	No	No	Significant only if this is the lone module
required	No	No	Failure eventually causes the stack to fail
requisite	Yes	No	Same as required, but fails stack immediately
sufficient	No	Yes	The name is kind of a lie; see comments below

a. Linux and Solaris only for include, Solaris only for binding

required modules are required to succeed; a failure of any one of them guarantees that the stack as a whole will eventually fail. However, the failure of a module that is marked required doesn't immediately stop execution of the stack. If you want that behavior, you need to use the requisite control flag instead of required.

The success of a sufficient module aborts the stack immediately. However, the ultimate result of the stack isn't guaranteed to be success because sufficient modules can't override the failure of earlier required modules. If an earlier required module has already failed, a successful sufficient module aborts the stack *and* returns failure as the overall result. Solaris's binding flag acts like sufficient, but failure of the binding module ensures eventual failure of the stack. By contrast, failure of a sufficient module is treated like the failure of an optional module: it makes no difference to the final result unless it is the only module in the stack.

Clear as mud, hmm? To make things even more complicated, Linux has a parallel system of alternative control flags that you can theoretically use instead of these cross-system standards. Overall, the *control-flag* system would take another page or two to really explain in detail. We don't do that here, however, because PAM configurations tend to be relatively stereotyped; you're unlikely to be writing your own from scratch. We mention some of the details only to impress upon you that the control flags don't have the straightforward meanings their names might suggest. If you're going to modify your systems' security settings, make sure that you understand the system thoroughly and that you double-check the particulars. (You won't configure PAM every day. How long will you remember which version is requisite and which is required?)

For easy reference, here's another copy of that same Solaris **pam.conf** example:

```
# login service

login   auth  requisite       pam_authtok_get.so.1
login   auth  required        pam_dhkeys.so.1
login   auth  required        pam_unix_cred.so.1
login   auth  required        pam_unix_auth.so.1
login   auth  required        pam_dial_auth.so.1
...
```

Let's look at the specific modules.

The pam_authtok_get library routine prompts the user for a login name (if one has not already been set) and password and stores these values in the authentication token called PAM_AUTHTOK. The pam_dhkeys module is used for RPC (remote procedure call) authentication for NIS or NIS+ and is looking for Diffie-Hellman keys, hence the name.

The pam_unix_cred module sets the credentials for the authenticated user, and the pam_unix_auth module performs the actual authentication, checking that the value stored in PAM_AUTHTOK is the user's correct password. Finally, the module pam_dial_auth authenticates the user for dialup access according to the contents of **/etc/dialups** and **/etc/d_passwd**.

You may see the same code module referred to more than once in a configuration file, with different *module-type* values. That's fine; multiple type implementations are often collected into a single library if they share significant code.

A detailed Linux configuration example

Linux moves each set of PAM configuration lines that refer to the same *service* into a separate file named after that service. The format is otherwise the same except that the *service* field is no longer needed.

For example, the **/etc/pam.d/login** file from a SUSE system is reproduced below with the included files expanded to form a more coherent example.

```
auth        requisite     pam_nologin.so
auth        [user_unknown=ignore success=ok ignore=ignore auth_err=die
    default=bad] pam_securetty.so
auth        required      pam_env.so
auth        required      pam_unix2.so
account     required      pam_unix2.so
password    requisite     pam_pwcheck.so nullok cracklib
password    required      pam_unix2.so use_authtok nullok
session     required      pam_loginuid.so
session     required      pam_limits.so
session     required      pam_unix2.so
session     optional      pam_umask.so
session     required      pam_lastlog.so nowtmp
session     optional      pam_mail.so standard
session     optional      pam_ck_connector.so
```

The auth stack includes several modules. On the first line, the pam_nologin module checks for the existence of the **/etc/nologin** file. If it exists, the module aborts the login immediately unless the user is root. The pam_securetty module ensures that root can only log in on terminals listed in **/etc/securetty**. This line uses the alternative Linux syntax described in the **pam.conf** man page. In this case, the requested behavior is similar to that of the required control flag. pam_env sets environment variables from **/etc/security/pam_env.conf**, and

finally, pam_unix2 checks the user's credentials by performing standard UNIX authentication. If any of these modules fail, the auth stack returns an error.

The account stack includes only the pam_unix2 module, which in this context assesses the validity of the account itself. It returns an error if, for example, the account has expired or the password must be changed. In the latter case, the module collects a new password from the user and passes it to the password modules.

The pam_pwcheck line checks the strength of proposed new passwords by calling the **cracklib** library. It returns an error if the new password does not meet the requirements. However, it also allows empty passwords because of the nullok flag. The pam_unix2 line updates the actual password.

Finally, the session modules perform several housekeeping chores. pam_loginuid sets the kernel's loginuid process attribute to the user's UID. pam_limits reads resource usage limits from **/etc/security/limits.conf** and sets the corresponding process parameters that enforce them. pam_unix2 logs the user's access to the system, and pam_umask sets an initial file creation mode. The pam_lastlog module displays the user's last login time as a security check, and the pam_mail module prints a note if the user has new mail. Finally, pam_ck_connector notifies the **ConsoleKit** daemon (a system-wide daemon that manages login sessions) of the new login.

At the end of the process, the user has been successfully authenticated and PAM returns control to **login**.

22.6 SETUID PROGRAMS

Setuid programs (executables on which the setuid bit has been set) run as the user that owns the executable file. For example, the **passwd** program must run as root in order to modify the **/etc/shadow** file when users change their passwords. See *Setuid and setgid execution* on page 106 for basic information about this feature.

Programs that run setuid, especially ones that run setuid to root, are prone to security problems. The setuid commands distributed with the system are theoretically secure; however, security holes have been discovered in the past and will undoubtedly be discovered in the future.

The surest way to minimize the number of setuid *problems* is to minimize the number of setuid *programs*. Think twice before installing software that needs to run setuid, and avoid using the setuid facility in your own home-grown software. *Never* use setuid execution on programs that were not explicitly written with setuid execution in mind.

You can disable setuid and setgid execution on individual filesystems by specifying the **nosuid** option to **mount**. It's a good idea to use this option on filesystems that contain users' home directories or that are mounted from less trustworthy administrative domains.

It's useful to scan your disks periodically to look for new setuid programs. A hacker who has breached the security of your system sometimes creates a private setuid shell or utility to facilitate repeat visits. Some of the tools discussed starting on page 914 locate such files, but you can do just as well with **find**. For example, the one-liner script

```
/usr/bin/find / -user root -perm -4000 -print |
    /bin/mail -s "Setuid root files" netadmin
```

mails a list of all files that are setuid to root to the "netadmin" user. (In practice, you may need to be more specific about which filesystems to search.)

22.7 EFFECTIVE USE OF CHROOT

The **chroot** system call confines a process to a specific directory. It disallows access to files outside or above that directory and thereby limits the damage the process can cause if it should be compromised by a hacker.

The **chroot** command is a simple wrapper around this system call. In addition, some security-sensitive daemons have **chroot** support built in and need only have this mode turned on in their configuration files.

Security experts sometimes frown upon use of **chroot** for security purposes because they believe that when it is poorly used or misunderstood, it can give administrators a false sense of security. They complain that some administrators use **chroot** to excuse themselves from other forms of security diligence such as regular software updates and close security monitoring.

These points are not inaccurate, but they're not the last word on **chroot**, either. Similar claims could be made regarding network firewalls, but few experts would recommend removing the packet filter from your network. Used correctly and as a supplemental layer of protection, **chroot** is a worthy addition to your security arsenal (even if that was not the feature's original design intent).

The following scenarios illustrate reasonable uses of **chroot**:

- You want to run a non-root daemon process such as Apache or BIND within a restricted filesystem subtree. If the daemon is compromised, the attacker will be restricted to the subtree as long as no privilege escalation vulnerabilities exist.
- You want to restrict remote users to a specific set of files and commands.

However, **chroot** can only protect you in these scenarios if all of the following conditions are met:

- All processes in the **chroot** jail run without root privileges. Processes that run as root always have the ability to break out of the **chroot** jail.
- You are not using setuid root execution within the jail.

- The **chroot** environment is up to date and minimal, in the sense that it contains only the executables, libraries, and configuration files that are needed to support the intended task.

In this era of shared libraries and interprocess dependencies, constructing a proper jail cell can be tricky. The JailKit (olivier.sessink.nl/jailkit) includes several scripts to help you create **chroot**ed environments.

22.8 SECURITY POWER TOOLS

Some of the time-consuming chores mentioned in the previous sections can be automated with freely available tools. Here are a few of the tools you'll want to look at.

Nmap: network port scanner

Nmap is a network port scanner. Its main function is to check a set of target hosts to see which TCP and UDP ports have servers listening on them.[3] Since most network services are associated with "well known" port numbers, this information tells you quite a lot about the software a machine is running.

Running Nmap is a great way to find out what a system looks like to someone on the outside who is trying to break in. For example, here's a report from a production Ubuntu system:

```
ubuntu$ nmap -sT ubuntu.booklab.atrust.com

Starting Nmap 4.20 ( http://insecure.org ) at 2009-11-01 12:31 MST
Interesting ports on ubuntu.booklab.atrust.com (192.168.20.25):
Not shown: 1691 closed ports
PORT     STATE    SERVICE
25/tcp   open     smtp
80/tcp   open     http
111/tcp  open     rpcbind
139/tcp  open     netbios-ssn
445/tcp  open     microsoft-ds
3306/tcp open     mysql

Nmap finished: 1 IP address (1 host up) scanned in 0.186 seconds
```

By default, **nmap** includes the **-sT** argument to try to connect to each TCP port on the target host in the normal way.[4] Once a connection has been established, **nmap** immediately disconnects, which is impolite but not harmful to a properly written network server.

3. As described in Chapter 14, a port is a numbered communication channel. An IP address identifies an entire machine, and an IP address + port number identifies a specific server or network conversation on that machine.
4. Actually, only the privileged ports (those with port numbers under 1,024) and the well-known ports are checked by default. Use the **-p** option to explicitly specify the range of ports to scan.

From the example above, we can see that the host ubuntu is running two services that are likely to be unused and that have historically been associated with security problems: **portmap** (rpcbind) and an email server (smtp). An attacker would most likely probe those ports for more information as a next step in the information-gathering process.

The STATE column in **nmap**'s output shows open for ports that have servers listening, closed for ports with no server, unfiltered for ports in an unknown state, and filtered for ports that cannot be probed because of an intervening packet filter. **nmap** does not classify ports as unfiltered unless it is running an ACK scan. Here are results from a more secure server, secure.booklab.atrust.com:

```
ubuntu$ nmap -sT secure.booklab.atrust.com

Starting Nmap 4.20 ( http://insecure.org ) at 2009-11-01 12:42 MST
Interesting ports on secure.booklab.atrust.com (192.168.20.35):
Not shown: 1691 closed ports
PORT      STATE SERVICE
25/tcp    open  smtp
80/tcp    open  http

Nmap finished: 1 IP address (1 host up) scanned in 0.143 seconds
```

In this case, it's clear that the host is set up to allow SMTP (email) and an HTTP server. A firewall blocks access to other ports.

In addition to straightforward TCP and UDP probes, **nmap** also has a repertoire of sneaky ways to probe ports without initiating an actual connection. In most cases, **nmap** probes with packets that look like they come from the middle of a TCP conversation (rather than the beginning) and waits for diagnostic packets to be sent back. These stealth probes may be effective at getting past a firewall or at avoiding detection by a network security monitor on the lookout for port scanners. If your site uses a firewall (see *Firewalls* on page 932), it's a good idea to probe it with these alternative scanning modes to see what they turn up.

nmap has the magical and useful ability to guess what operating system a remote system is running by looking at the particulars of its implementation of TCP/IP. It can sometimes even identify the software that's running on an open port. The **-O** and **-sV** options, respectively, turn on this behavior. For example:

```
ubuntu$ sudo nmap -sV -O secure.booklab.atrust.com

Starting Nmap 4.20 ( http://insecure.org ) at 2009-11-01 12:44 MST
Interesting ports on secure.booklab.atrust.com (192.168.20.35):
Not shown: 1691 closed ports
PORT      STATE    SERVICE   VERSION
25/tcp    open     smtp      Postfix smtpd
80/tcp    open     http      lighttpd 1.4.13
Device type: general purpose
Running: Linux 2.4.X|2.5.X|2.6.X
OS details: Linux 2.6.16 - 2.6.24
Nmap finished: 1 IP address (1 host up) scanned in 8.095 seconds
```

This feature can be very useful for taking an inventory of a local network. Unfortunately, it is also very useful to hackers, who can base their attacks on known weaknesses of the target OSes and servers.

Keep in mind that most administrators don't appreciate your efforts to scan their network and point out its vulnerabilities, however well intended your motive. Do not run **nmap** on someone else's network without permission from one of that network's administrators.

Nessus: next-generation network scanner

Nessus, originally released by Renaud Deraison in 1998, is a powerful and useful software vulnerability scanner. At this point, it uses more than 31,000 plug-ins to check for both local and remote security flaws. Although it is now a closed source, proprietary product, it is still freely available, and new plug-ins are released regularly. It is the most widely accepted and complete vulnerability scanner available.

Nessus prides itself on being the security scanner that takes nothing for granted. Instead of assuming that all web servers run on port 80, for instance, it scans for web servers running on any port and checks them for vulnerabilities. Instead of relying on the version numbers reported by the service it has connected to, Nessus attempts to exploit known vulnerabilities to see if the service is susceptible.

Although a substantial amount of setup time is required to get Nessus running (it requires several packages that aren't installed on a typical system), it's well worth the effort. The Nessus system includes a client and a server. The server acts as a database and the client handles the GUI presentation. Nessus servers and clients exist for both Windows and UNIX platforms.

One of the great advantages of Nessus is the system's modular design, which makes it easy for third parties to add new security checks. Thanks to an active user community, Nessus is likely to be a useful tool for years to come.

John the Ripper: finder of insecure passwords

One way to thwart poor password choices is to try to break the passwords yourself and to force users to change passwords that you have broken. John the Ripper is a sophisticated tool by Solar Designer that implements various password-cracking algorithms in a single tool. It replaces the tool **crack**, which was covered in previous editions of this book.

Even though most systems use a shadow password file to hide encrypted passwords from public view, it's still wise to verify that your users' passwords are crack resistant.[5] Knowing a user's password can be useful because people tend to use the same password over and over again. A single password might provide access to another system, decrypt files stored in a user's home directory, and allow access to

5. Especially the passwords of system administrators who have **sudo** privileges

financial accounts on the web. (Needless to say, it's not very security-smart to reuse a password this way. But nobody wants to remember ten passwords.)

Considering its internal complexity, John the Ripper is an extremely simple program to use. Direct **john** to the file to be cracked, most often **/etc/shadow**, and watch the magic happen:

```
$ sudo ./john /etc/shadow
Loaded 25 password hashes with 25 different salts (FreeBSD MD5 [32/32])
password   (jsmith)
badpass    (tjones)
```

In this example, 25 unique passwords were read from the shadow file. As passwords are cracked, John prints them to the screen and saves them to a file called **john.pot**. The output contains the password in the left column with the login in parentheses in the right column. To reprint passwords after **john** has completed, run the same command with the **-show** argument.

As of this writing, the most recent stable version of John the Ripper is 1.7.3.4. It's available from openwall.com/john. Since John the Ripper's output contains the passwords it has broken, you should carefully protect the output and delete it as soon as you are done checking to see which users' passwords are insecure.

As with most security monitoring techniques, it's important to obtain explicit management approval before cracking passwords with John the Ripper.

hosts_access: host access control

Network firewalls are a first line of defense against access by unauthorized hosts, but they shouldn't be the only barrier in place. Two files, **/etc/hosts.allow** and **/etc/hosts.deny**, also referred to as TCP wrappers, can restrict access to services according to the origin of network requests. The **hosts.allow** file lists the hosts that are allowed to connect to a specific service, and the **hosts.deny** file restricts access. However, these files control access only for services that are **hosts_access** aware, such as those managed by **inetd**, **xinetd**, **sshd**, and some configurations of **sendmail**.

In most cases it is wise to be restrictive and permit access only to essential services from designated hosts. We suggest denying access by default in the **hosts.deny** file with the single line

```
ALL:ALL
```

You can then permit access on a case-by-case basis in **hosts.allow**. The following configuration allows access to SSH from hosts on the 192.168/16 networks and to **sendmail** from anywhere.

```
sshd: 192.168.0.0/255.255.0.0
sendmail: ALL
```

The format of an entry in either file is *service: host* or *service: network*. Failed connection attempts are noted in syslog. Connections from hosts that are not permitted to access the service are immediately closed.

Most Linux distributions include **hosts.allow** and **hosts.deny** files by default, but they're usually empty. Our other example systems all offer TCP wrappers as an option after installation.

Bro: the programmable network intrusion detection system

Bro is an open source network intrusion detection system (NIDS) that monitors network traffic and looks for suspicious activity. It was originally written by Vern Paxson and is available from bro-ids.org.

Bro inspects all traffic flowing into and out of a network. It can operate in passive mode, in which it generates alerts for suspicious activity, or in active mode, in which it injects traffic to disrupt malicious activity. Both modes likely require modification of your site's network configuration.

Unlike other NIDSs, Bro monitors traffic flows rather than just matching patterns inside individual packets. This method of operation means that Bro can detect suspicious activity based on who talks to whom, even without matching any particular string or pattern. For example, Bro can

- Detect systems used as "stepping stones" by correlating inbound and outbound traffic
- Detect a server that has a back door installed by watching for unexpected outbound connections immediately after an inbound one
- Detect protocols running on nonstandard ports
- Report correctly guessed passwords (and ignore the incorrect guesses)

Some of these features require substantial system resources, but Bro includes clustering support to help you manage a group of sensor machines.

The configuration language for Bro is complex and requires significant coding experience to use. Unfortunately, there is no simple default configuration for a novice to install. Most sites require a moderate level of customization.

Bro is supported to some extent by the Networking Research Group of the International Computer Science Institute (ICSI), but mostly it's maintained by the community of Bro users. If you are looking for a turnkey commercial NIDS, you will probably be disappointed by Bro. However, Bro can do things that no commercial NIDS can do, and it can either supplement or replace a commercial solution in your network.

Snort: the popular network intrusion detection system

Snort (snort.org) is an open source network intrusion prevention and detection system originally written by Marty Roesch and now maintained by Sourcefire, a

commercial entity. It has become the de facto standard for home-grown NIDS deployments and is also the basis of many commercial and "managed services" NIDS implementations.

Snort itself is distributed for free as an open source package. However, Sourcefire charges a subscription fee for access to the most recent set of detection rules.

A number of third-party platforms incorporate or extend Snort, and some of those projects are open source. One excellent example is Aanval (aanval.com), which aggregates data from multiple Snort sensors in a web-based console.

Snort captures raw packets off the network wire and compares them with a set of rules, aka signatures. When Snort detects an event that's been defined as interesting, it can alert a system administrator or contact a network device to block the undesired traffic, among other actions.

Although Bro is a much more powerful system, Snort is a lot simpler and easier to configure, attributes that make it a good choice as a "starter" NIDS platform.

OSSEC: host-based intrusion detection

Do you lie awake at night wondering if the security of your systems has been breached? Do you think a disgruntled coworker might be installing malicious programs on your systems? If you answered yes to either of these questions, you may want to consider installing a host-based intrusion detection system (HIDS) such as OSSEC.

OSSEC is free software and is available as source code under the GNU General Public License. Commercial support is available from Third Brigade (recently acquired by Trend Micro). OSSEC is available for Linux, Solaris, HP-UX, AIX, and Windows. It provides the following services:

- Rootkit detection
- Filesystem integrity checks
- Log file analysis
- Time-based alerting
- Active responses

OSSEC runs on the systems of interest and monitors their activity. It can send alerts or take action according to a set of rules that you configure. For example, OSSEC can monitor systems for the addition of unauthorized files and send email notifications like this one:

```
Subject: OSSEC Notification - courtesy - Alert level 7
Date: Fri, 15 Jan 2010 14:53:04 -0700
From: OSSEC HIDS <ossecm@courtesy.atrust.com>
To: <courtesy-admin@atrust.com>

OSSEC HIDS Notification.
2010 Jan 15 14:52:52
```

```
Received From: courtesy->syscheck
Rule: 554 fired (level 7) -> "File added to the system."
Portion of the log(s):

New file
'/courtesy/httpd/barkingseal.com/html/wp-content/uploads/2010/01/hbird.jpg'
added to the file system.

--END OF NOTIFICATION
```

In this way, OSSEC acts as your 24/7 eyes and ears on the system. We recommend running OSSEC on every production system, in combination with a change management policy (discussed in Chapter 32, *Management, Policy, and Politics*, on page 1211).

OSSEC basic concepts

OSSEC has two primary components: the manager (server) and the agents (clients). You need one manager on your network, and you should install that component first. The manager stores the file-integrity-checking databases, logs, events, rules, decoders, major configuration options, and system auditing entries for the entire network. A manager can connect to any OSSEC agent, regardless of its operating system. The manager can also monitor certain devices that do not have a dedicated OSSEC agent.

Agents run on the systems you want to monitor and report back to the manager. By design, they have a small footprint and operate with a minimal set of privileges. Most of the agent's configuration is obtained from the manager. Communication between the server and the agent is encrypted and authenticated. You need to create an authentication key for each agent on the manager.

OSSEC classifies alerts by severity at levels 0 to 15; 15 is the highest severity.

OSSEC installation

OSSEC is not yet part of the major UNIX and Linux distributions, even as a fetchable package. Therefore, you will need to download the source code package with a web browser or a tool such as **wget** and then build the software:

```
$ wget http://ossec.net/files/ossec-hids-latest.tar.gz
$ tar -zxvf ossec-hids-latest.tar.gz
$ cd ossec-hids-*
$ sudo ./install.sh
```

The install script asks what language you prefer (use "en" for English), and then what type of installation you want to perform: server, agent, or local. If you are only installing OSSEC on a single, personally managed system, you may want to choose local. Otherwise, first do the server install on the system you want to be your OSSEC manager, and then install the agent on that and all other systems you want to monitor. The install script asks some additional questions, too, such as to what email address alerts should be sent and which monitoring modules should be enabled.

Once the installation has finished, start OSSEC with

```
server$ sudo /var/ossec/bin/ossec-control start
```

Next, register each agent with the manager. On the server, run

```
server$ sudo /var/ossec/bin/manage_agents
```

You'll see a menu that looks something like this:

```
****************************************
* OSSEC HIDS v2.3 Agent manager.
* The following options are available:
****************************************
   (A)dd an agent (A).
   (E)xtract key for an agent (E).
   (L)ist already added agents (L).
   (R)emove an agent (R).
   (Q)uit.
Choose your action: A,E,L,R or Q:
```

Select option **A** to add an agent, and then type in the name and IP address of the agent. Next, select option **E** to extract the agent's key. Here's what that looks like:

```
Available agents:
   ID: 001, Name: linuxclient1, IP: 192.168.74.3
Provide the ID of the agent to extract the key (or '\q' to quit): 001
Agent key information for '001' is:
MDAyIGxpbnV4Y2xpZW50MSAxOTIuMTY4Ljc0LjMgZjk4YjMyYzlkMjg5MWJlMT
…
```

Finally, log in to the agent system and run **manage_agents** there:

```
agent$ sudo /var/ossec/bin/manage_agents
```

On the client, you will see that the menu has somewhat different options.

```
****************************************
* OSSEC HIDS v2.3 Agent manager.
* The following options are available:
****************************************
   (I)mport key from the server (I).
   (Q)uit.
Choose your action: I or Q:
```

Select option **I** and then cut and paste the key you extracted above. After you have added an agent, you must restart the OSSEC server. Repeat the process of key generation, extraction, and installation for each agent you want to connect.

OSSEC configuration

Once OSSEC is installed and running, you'll want to tweak it so that it gives you just enough information, but not too much. The majority of the configuration is stored on the server in the **/var/ossec/etc/ossec.conf** file. This XML-style file is well commented and fairly intuitive, but it contains dozens of options.

A common item you may want to configure is the list of files to ignore when doing file integrity (change) checking. For example, if you have a custom application that writes its log file to **/var/log/customapp.log**, you can add the following line to the <syscheck> section of the file:

```
<syscheck>
<ignore>/var/log/customapp.log</ignore>
</syscheck>
```

After you've made this change and restarted the OSSEC server, OSSEC will stop alerting you every time the log file changes. The many OSSEC configuration options are documented at ossec.net/main/manual/configuration-options.

It takes time and effort to get any HIDS system running and tuned. But after a few weeks, you'll have filtered out the noise and the system will start to provide valuable information about changing conditions in your environment.

22.9 MANDATORY ACCESS CONTROL (MAC)

Mandatory Access Control is an alternative to the traditional UNIX access control system that vests control of all permissions in the hands of a security administrator. In contrast to the standard model (described in Chapter 4, *Access Control and Rootly Powers*, and to some extent in Chapter 6, *The Filesystem*), MAC does not allow users to modify any permissions, even on their own objects.

MAC security policies control access according to the perceived sensitivity of the resource being controlled. Users are assigned a security classification from a structured hierarchy. Users can read and write items at the same classification level or lower but cannot access items at a higher classification. For example, a user with "secret" access can read and write "secret" objects but cannot read objects that are classified as "top secret."

A well-implemented MAC policy relies on the principle of least privilege (allowing access only when necessary), much as a properly designed firewall allows only specifically recognized services and clients to pass. MAC can prevent software with code execution vulnerabilities (e.g., buffer overflows) from compromising the system by limiting the scope of the breach to the few specific resources required by that software.

Needless to say, kernel modifications are necessary to implement MAC on UNIX and Linux. Our example UNIX systems (Solaris, HP-UX, and AIX) all are available in MAC-enabled versions at additional cost. These versions are called Solaris Trusted Extensions (formerly Trusted Solaris), HP-UX Security Containment, and Trusted AIX, respectively.

Unless you're handling sensitive data for a government entity, it is unlikely that you will ever need or encounter these security-enhanced editions.

Security-enhanced Linux (SELinux)

SELinux implements MAC for Linux systems. Although it has gained a foothold in a few distributions, it is notoriously difficult to administer and troubleshoot. This unattributed quote from a former version of the SELinux Wikipedia page vents the frustration felt by many sysadmins:

> *"Intriguingly, although the stated raison d'être of SELinux is to facilitate the creation of individualized access control policies specifically attuned to organizational data custodianship practices and rules, the supportive software tools are so sparse and unfriendly that the vendors survive chiefly on 'consulting,' which typically takes the form of incremental modifications to boilerplate security policies."*

Despite the administrative complexity of SELinux, its adoption has been slowly growing, particularly in environments, such as government agencies, with strict security requirements. Of our example Linux distributions, Red Hat Enterprise Linux has the most mature SELinux model. SELinux is available as an optional package for Ubuntu and SUSE.

Policy development is a complicated topic. To protect a new daemon, for example, a policy must carefully enumerate all the files, directories, and other objects to which the process needs access. For complicated software like **sendmail** or the Apache **httpd**, this task can be quite complex. At least one company offers a 3-day class on policy development.

Fortunately, many general policies are available on-line, and most distributions come with reasonable defaults. These can easily be installed and configured for your particular environment. A full-blown policy editor that aims to ease policy application can be found at seedit.sourceforge.net.

SELinux has been present in Red Hat Enterprise Linux since version 4. A default installation of RHEL enables SELinux protection out of the box.

/etc/selinux/config controls the SELinux configuration. The interesting lines are

```
SELINUX=enforcing
SELINUXTYPE=targeted
```

The first line has three possible values: `enforcing`, `permissive`, or `disabled`. The `enforcing` setting ensures that the loaded policy is applied and prohibits violations. `permissive` allows violations to occur but logs them through syslog, which is valuable for debugging. `disabled` turns off SELinux entirely.

`SELINUXTYPE` refers to the type of policy to be applied. Red Hat has two policies: `targeted`, which defines additional security for daemons that Red Hat has protected,[6] and `strict`, which protects the entire system. Although the `strict` policy is available, it is not supported by Red Hat; the restrictions are so tight that the

6. The protected daemons are **httpd**, **dhcpd**, **mailman**, **named**, **portmap**, **nscd**, **ntpd**, **mysqld**, **postgres**, **squid**, **winbindd**, and **ypbind**.

system is difficult to use. The targeted policy offers protection for important network daemons without affecting general system use, at least in theory. But even the targeted policy isn't perfect. If you're having problems with newly installed software, check **/var/log/messages** for SELinux errors.

SUSE uses Novell's implementation of MAC, called AppArmor. However, as of version 11.1, SUSE also includes basic SELinux functionality.

Ubuntu ships with AppArmor by default. SELinux packages are maintained for Ubuntu by Russell Coker, the Red Hat bloke who generated the targeted and strict policies.

22.10 CRYPTOGRAPHIC SECURITY TOOLS

Many of the UNIX protocols in common use date from a time before the wide deployment of the Internet and modern cryptography. Security was simply not a factor in the design of many protocols; in others, security concerns were waved away with the transmission of a plaintext password or with a vague check to see if packets originated from a trusted host or port.

These protocols now find themselves operating in the shark-infested waters of large corporate LANs and the Internet, where, it must be assumed, all traffic is open to inspection. Not only that, but there is little to prevent anyone from actively interfering in network conversations. How can you be sure who you're really talking to?

Cryptography solves many of these problems. It has been possible for a long time to scramble messages so that an eavesdropper cannot decipher them, but this is just the beginning of the wonders of cryptography. Developments such as public key cryptography and secure hashing have promoted the design of cryptosystems that meet almost any conceivable need.

An excellent resource for those interested in cryptography is RSA Laboratories' *Frequently Asked Questions about Today's Cryptography*, available for free from rsa.com/rsalabs. Despite the name, it is a book-length treatise downloadable in PDF format. The document hasn't been updated since 2000, but most of the information remains valid. Additionally, Stephen Levy's book *Crypto* is a comprehensive guide to the history of cryptography.

Kerberos: a unified approach to network security

The Kerberos system, designed at MIT, attempts to address some of the issues of network security in a consistent and extensible way. Kerberos is an authentication system, a facility that "guarantees" that users and services are in fact who they claim to be. It does not provide any additional security or encryption beyond that.

Kerberos uses DES to construct nested sets of credentials called "tickets." Tickets are passed around the network to certify your identity and to give you access to network services. Each Kerberos site must maintain at least one physically secure

machine (called the authentication server) to run the Kerberos daemon. This daemon issues tickets to users or services that present credentials, such as passwords, when they request authentication.

In essence, Kerberos improves upon traditional password security in only two ways: it never transmits unencrypted passwords on the network, and it relieves users from having to type passwords repeatedly, making password protection of network services somewhat more palatable.

The Kerberos community boasts one of the most lucid and enjoyable documents ever written about a cryptosystem, Bill Bryant's "Designing an Authentication System: a Dialogue in Four Scenes." It's required reading for anyone interested in cryptography and is available at

web.mit.edu/kerberos/www/dialogue.html

Kerberos offers a better network security model than the "ignoring network security entirely" model, but it is neither perfectly secure nor painless to install and run. It does not supersede the other security measures described in this chapter.

Unfortunately (and perhaps predictably), the Kerberos system distributed as part of Windows' Active Directory uses proprietary, undocumented extensions to the protocols. As a result, it does not interoperate well with distributions based on the MIT code. Fortunately, the **winbind** module lets UNIX and Linux systems interact with Active Directory's version of Kerberos. See *Configuring Kerberos for Active Directory integration* on page 1156 for more information.

PGP: Pretty Good Privacy

See page 763 for more information about email privacy.

Phil Zimmermann's PGP package provides a tool chest of bread-and-butter cryptographic utilities focused primarily on email security. It can be used to encrypt data, to generate signatures, and to verify the origin of files and messages.

PGP has an interesting history that includes lawsuits, criminal prosecutions, and the privatization of portions of the original PGP suite. Currently, PGP's file formats and protocols are being standardized by the IETF under the name OpenPGP, and multiple implementations of the proposed standard exist. The GNU project provides an excellent, free, and widely used implementation known as GnuPG at gnupg.org. For clarity, we refer to the system collectively as PGP even though individual implementations have their own names.

PGP is perhaps the most popular cryptographic software in common use. Unfortunately, the UNIX/Linux version is nuts-and-bolts enough that you have to understand a fair amount of cryptographic background in order to use it. Although you may find PGP useful in your own work, we don't recommend that you support it for users because it has been known to spark many puzzled questions. We have found the Windows version to be considerably easier to use than the **gpg** command with its 52 different operating modes.

Software packages on the Internet are often distributed with a PGP signature file that purports to guarantee the origin and purity of the software. However, it is difficult for people who are not die-hard PGP users to validate these signatures—not because the process is complicated, but because true security can only come from having collected a personal library of public keys from people whose identities you have directly verified. Downloading a single public key along with a signature file and software distribution is approximately as secure as downloading the distribution alone.

Some email clients, such as Mozilla Thunderbird, have add-ons that provide a simple GUI for encrypted incoming and outgoing messages. Enigmail, the solution for Thunderbird, can even search on-line public key databases if the key for your recipient isn't already in your key ring. See enigmail.mozdev.org for details.

SSH: the secure shell

The SSH system, written by Tatu Ylönen, is a secure replacement for **rlogin**, **rcp**, and **telnet**. It uses cryptographic authentication to confirm a user's identity and encrypts all communications between the two hosts. The protocol used by SSH is designed to withstand a wide variety of potential attacks. The protocol is documented by RFCs 4250 through 4256 and is now a proposed IETF standard.

SSH has morphed from being a freely distributed open source project (SSH1) to being a commercial product that uses a slightly different (and more secure) protocol, SSH2. Fortunately, the open source community has responded by releasing the excellent OpenSSH package (maintained by OpenBSD), which now implements both protocols.

The main components of SSH are a server daemon, **sshd**, and a few user-level commands, notably **ssh** for remote logins and **sftp/scp** for copying files. Other components are an **ssh-keygen** command that generates public key pairs and a couple of utilities that help support secure X Windows.

sshd can authenticate user logins in several different ways. It's up to you as the administrator to decide which of these methods are acceptable:

- **Method A:** If the name of the remote host from which the user is logging in is listed in **~/.rhosts**, **~/.shosts**, **/etc/hosts.equiv**, or **/etc/shosts.equiv**, then **sshd** logs in the user automatically without a password check. This scheme mirrors that of the old **rlogin** daemon and in our opinion is *never* acceptable for normal use.
- **Method B:** As a refinement of method A, **sshd** can also use public key cryptography to verify the identity of the remote host. For that to happen, the remote host's public key (generated at install time) must be listed in the local host's **/etc/ssh_known_hosts** file or the user's **~/.ssh/known_hosts** file.

If the remote host can prove that it knows the corresponding private key (normally stored in **/etc/ssh_host_key**, a world-unreadable file), then **sshd** logs in the user without asking for a password.

Method B is more restrictive than method A, but we think it's still not quite secure enough. If the security of the originating host is compromised, the local site will be compromised as well.

- **Method C: sshd** can use public key cryptography to establish the user's identity. At login time, the user must have access to a copy of his or her private key file and must supply a password to decrypt it. The key can also be created without a password, which is a reasonable option for automating logins from remote systems.

  This method is the most secure, but it's annoying to set up. It also means that users cannot log in when traveling unless they bring along a copy of their private key file (perhaps on a USB key, hopefully encrypted).

  If you decide to use key pairs, make extensive use of **ssh -v** during the troubleshooting process.

- **Method D:** Finally, **sshd** can simply allow the user to enter his or her normal login password. This makes **ssh** behave very much like **telnet**, except that the password and session are both encrypted. The main drawbacks of this method are that system login passwords can be relatively weak if you have not beefed up their security, and that ready-made tools (such as John the Ripper) have been designed to break them. However, this method is probably the best choice for normal use.

Authentication policy is set in **/etc/sshd_config**. This file gets filled up with configuration rubbish for you as part of the installation process, but you can safely ignore most of it. The options relevant to authentication are shown in Table 22.2.

Table 22.2 Authentication-related options in /etc/sshd_config

Option	Meth[a]	Dflt	Meaning when turned on
RhostsAuthentication	A	No	Obeys **~/.shosts**, **/etc/shosts.equiv**, etc.
RhostsRSAAuthentication	B	Yes	Allows **~/.shosts** et al., but requires host key
IgnoreRhosts	A,B	No	Ignores the **~/.rhosts** and **hosts.equiv** files[b]
IgnoreRootRhosts	A,B	No[c]	Prevents **rhosts**/**shosts** authentication for root
RSAAuthentication	C	Yes	Allows per-user public key authentication
PasswordAuthentication	D	Yes	Allows use of normal login password

a. The authentication methods to which this variable is relevant
b. But continues to honor **~/.shosts** and **shosts.equiv**
c. Defaults to the value of IgnoreRhosts

Our suggested configuration, which allows methods C and D but not methods A or B, is as follows:

```
RhostsAuthentication no
RhostsRSAAuthentication no
RSAAuthentication yes
PasswordAuthentication yes
```

It is never wise to allow root to log in remotely. Superuser access should be achieved through the use of **sudo**. To encourage this behavior, use the option

```
PermitRootLogin no
```

The first time you connect to a new system through SSH, you are prompted to accept the remote host's public key (which is usually generated as part of the server's installation of OpenSSH, or soon thereafter). A truly paranoid user might manually verify it, but most of us blindly accept the key, which is then stored in the **~/.ssh/known_hosts** file for future use. SSH won't mention the server's key again unless it changes. Unfortunately, users' rubber-stamping the keys of new systems leaves you vulnerable to a man-in-the-middle attack if the host key was actually being presented by an attacker's system.

A DNS record known as SSHFP has been developed to address this vulnerability. The premise is that the server's key is stored as a DNS record. When a client connects to an unknown system, SSH looks up the SSHFP record to verify the server's key rather than asking the user to verify it.

The **sshfp** utility, available from xelerance.com/software/sshfp, generates SSHFP DNS resource records either by scanning a remote server or by parsing a previously accepted key from the **known_hosts** file. (Of course, either choice assumes that the source of the key is known to be correct.) Usage is quite simple: use the **-s** flag to generate a key from a network scan, or use **-k** to scan the **known_hosts** file (the default). For example, the following command generates a BIND-compatible SSHFP record for solaris.booklab.atrust.com:

```
solaris$ sshfp solaris.booklab.atrust.com
solaris.booklab.atrust.com   IN  SSHFP 1 1 94a26278ee713a37f6a78110f1ad9bd…
solaris.booklab.atrust.com   IN  SSHFP 2 1 7cf72d02e3d3fa947712bc56fd0e0a3i…
```

Add these records to the domain's zone file (be careful of the names and the $ORIGIN), reload the domain, and use **dig** to verify the key:

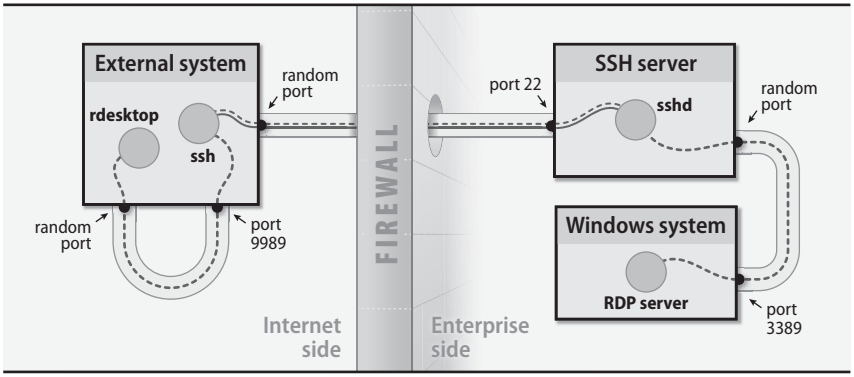
```
solaris$ dig solaris.booklab.atrust.com. IN SSHFP | grep SSHFP
; <<>> DiG 9.5.1-P2 <<>> solaris.booklab.atrust.com. IN SSHFP
; solaris.booklab.atrust.com. IN SSHFP
solaris.booklab.atrust.com.  38400  IN  SSHFP 1 1 94a26278ee713a37f6a78110f…
solaris.booklab.atrust.com.  38400  IN  SSHFP 2 1 7cf72d02e3d3fa947712bc56f…
```

ssh does not consult SSHFP records by default. Add the VerifyHostKeyDNS option to **/etc/ssh/ssh_config** to enable it. As with most SSH client options, you can

also pass **-o "VerifyHostKeyDNS yes"** on the **ssh** command line when first accessing a new system.

SSH has a couple of ancillary functions that are useful for system administrators. One of these is the ability to tunnel TCP connections securely through an encrypted SSH channel, thereby allowing connectivity to insecure or firewalled services at remote sites. This functionality is ubiquitous among SSH clients and is simple to configure. Exhibit A shows a typical use of an SSH tunnel and should help clarify how it works.

Exhibit A An SSH tunnel for RDP



In this scenario, a remote user wants to establish an RDP (remote desktop) connection to a Windows system on the enterprise network. Access to that host or to port 3389 is blocked by the firewall, but since the user has SSH access, he can route his connection through the SSH server.

To set this up, the user logs in to the remote SSH server with **ssh**. On the **ssh** command line, he specifies an arbitrary (but specific; in this case, 9989) local port that **ssh** should forward through the secure tunnel to the remote Windows machine's port 3389. (For the standard OpenSSH implementation, the option to request this behavior is simply **-L** *localport***:***remotehost***:***remoteport.)* All source ports in this example are marked as random since programs choose an arbitrary port from which they initiate connections.

To access the Windows machine's desktop, the user then opens the remote desktop client (here, **rdesktop**) and enters localhost:9989 as the address of the server to connect to. The local **ssh** receives the connection on port 9989 and tunnels the traffic over the existing connection to the remote **sshd**. In turn, **sshd** forwards the connection to the Windows host.

Of course, tunnels such as these can be intentional or unintentional back doors as well. System administrators should use tunnels with caution and should also watch for unauthorized misuse of this facility by users.

In recent years, SSH has become the target of regular brute-force password attacks. Attackers perform repeated authentication attempts as common users, such as root, joe, or admin. Evidence of the attacks can be seen in the logs as hundreds or thousands of failed logins. Disabling password authentication is the best protection against these attacks. For now, attackers seem to be focusing only on port 22, so moving your SSH server to another port is an effective countermeasure. But history shows that this type of "security through obscurity" is rarely effective for long. Running password checks on your systems can reveal weak passwords that are likely to be broken by brute-force attacks.

Stunnel

Stunnel, created by Michal Trojnara, is an open source package that encrypts arbitrary TCP connections, much in the manner of SSH. It uses SSL, the Secure Sockets Layer, to create end-to-end tunnels through which it passes data to and from an unencrypted service. It is known to work well with insecure services such as Telnet, IMAP, and POP.

A **stunnel** daemon runs on both the client and server systems. The local **stunnel** usually accepts connections on the service's traditional port (e.g., port 25 for SMTP) and routes them through SSL to a **stunnel** on the remote host. The remote **stunnel** accepts the connection, decrypts the incoming data, and routes it to the remote port on which the server is listening. This system allows unencrypted services to take advantage of the confidentiality and integrity offered by encryption without requiring any software changes. Client software need only be configured to look for services on the local system rather than on the server that will ultimately provide them.

The Telnet protocol makes a good example because it consists of a simple daemon listening on a single port. To **stunnel**ify a Telnet link, you first create an SSL certificate. Stunnel is SSL library independent, so any standards-based implementation will do; we like OpenSSL. To generate the certificate:

```
server$ sudo openssl req -new -x509 -days 365 -nodes -out stunnel.pem
    -keyout stunnel.pem
Generating a 1024 bit RSA private key
.++++++
.....................................++++++
writing new private key to 'stunnel.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
Country Name (2 letter code) [GB]:US
State or Province Name (full name) [Berkshire]:Colorado
Locality Name (eg, city) [Newbury]:Boulder
```

```
Organization Name (eg, company) [My Company Ltd]:Booklab, Inc.
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or server's hostname) []:server.example.com
Email Address []:
```

This command creates a self-signed, passphrase-less certificate. Although not using a passphrase is a convenience (a real human doesn't have to be present to type a passphrase each time **stunnel** restarts), it also introduces a security risk. Be careful to protect the certificate file with strong permissions.

Next, define the configuration for both the server and client **stunnel**s. The standard configuration file is **/etc/stunnel/stunnel.conf**, but you can create several configurations if you want to run more than one tunnel.

```
cert = /etc/stunnel/stunnel.pem
chroot = /var/run/stunnel/
pid = /stunnel.pid
setuid = nobody
setgid = nobody
debug = 7
output = /var/log/stunnel.log
client = no

[telnets]
accept = 992
connect = 23
```

There are a couple of important points to note about the server configuration. First, the chroot statement confines the **stunnel** process to the **/var/run/stunnel** directory. Paths for accessory files may need to be expressed in either the regular system namespace or the **chroot**ed namespace, depending on the point at which they are opened. Here, the **stunnel.pid** file is actually located in **/var/run/stunnel**.

The [telnets] section has two statements: accept tells **stunnel** to accept connections on port 992, and connect passes those connections through to port 23, the actual Telnet service.

The client configuration is similar:

```
cert = /etc/stunnel/stunnel.pem
chroot = /var/run/stunnel/
pid = /stunnel.pid
setuid = nobody
setgid = nobody
debug = 7
output = /var/log/stunnel.log
client = yes

[telnets]
accept = 23
connect = server.example.com:992
```

A couple of directives are reversed relative to the server configuration. The statement client = yes tells the program to initiate **stunnel** connections rather than accept them. The local **stunnel** listens for connections on port 23 and connects to the server on port 992. The hostname in the connect directive should match the entry specified when the certificate was created.

Both the client and the server **stunnel**s can be started with no command-line arguments. If you check with **netstat -an**, you should see the server **stunnel** waiting for connections on port 992 while the client **stunnel** waits on port 23.

To access the tunnel, a user simply **telnet**s to the local host:

```
client$ telnet localhost 23
Trying 127.0.0.1...
Connected to localhost (127.0.0.1).
Escape character is '^]'.
Red Hat Enterprise Linux WS release 4 (Nahant Update 2)
Kernel 2.6.9-5.EL on an i686
login:
```

The user can now safely log in without fear of password thievery. A vigilant administrator would be careful to use TCP wrappers to restrict connections on the client to only the local interface—the intent is not to allow the world to **telnet** securely to the server! **stunnel** is one of several programs that have built-in wrapper support and do not require the use of **tcpd** to restrict access. Visit stunnel.org for instructions.

22.11 FIREWALLS

In addition to protecting individual machines, you can also implement security precautions at the network level. The basic tool of network security is the firewall, a device or piece of software that prevents unwanted packets from accessing networks and systems. Firewalls are ubiquitous today and are found in devices ranging from desktop systems and servers to consumer routers and enterprise-grade network appliances.

Packet-filtering firewalls

A packet-filtering firewall limits the types of traffic that can pass through your Internet gateway (or through an internal gateway that separates domains within your organization) on the basis of information in the packet header. It's much like driving your car through a customs checkpoint at an international border crossing. You specify which destination addresses, port numbers, and protocol types are acceptable, and the gateway simply discards (and in some cases, logs) packets that don't meet the profile.

Packet-filtering software is included in Linux systems in the form of **iptables**, in Solaris and HP-UX as IPFilter, and in AIX as **genfilt**. See the details beginning on page 935 for more information.

Although these tools are capable of sophisticated filtering and bring a welcome extra dose of security, we generally discourage the use of UNIX and Linux systems as network routers and, most especially, as enterprise firewall routers. The complexity of general-purpose operating systems makes them inherently less secure and less reliable than task-specific devices. Dedicated firewall appliances such as those made by Check Point and Cisco are a better option for site-wide network protection.

How services are filtered

Most well-known services are associated with a network port in the **/etc/services** file or its vendor-specific equivalent. The daemons that provide these services bind to the appropriate ports and wait for connections from remote sites.[7] Most of the well-known service ports are "privileged," meaning that their port numbers are in the range 1 to 1023. These ports can only be used by a process running as root. Port numbers 1024 and higher are referred to as nonprivileged ports.

Service-specific filtering is based on the assumption that the client (the machine that initiates a TCP or UDP conversation) uses a nonprivileged port to contact a privileged port on the server. For example, if you wanted to allow only inbound SMTP connections to a machine with the address 192.108.21.200, you would install a filter that allowed TCP packets destined for port 25 at that address and that permitted outbound TCP packets from that address to anywhere.[8] The exact way that such a filter is installed depends on the kind of router or filtering system you are using.

*See page 977 for more information about setting up an **ftp** server.*

Some services, such as FTP, add a twist to the puzzle. The FTP protocol actually uses two TCP connections when transferring a file: one for commands and the other for data. The client initiates the command connection, and the server initiates the data connection[9]. Ergo, if you want to use FTP to retrieve files from the Internet, you must permit inbound access to all nonprivileged TCP ports since you have no idea what port might be used to form an incoming data connection.

This tweak largely defeats the purpose of packet filtering because some notoriously insecure services (for example, X11 at port 6000) naturally bind to nonprivileged ports. This configuration also creates an opportunity for curious users within your organization to start their own services (such as a **telnet** server at a nonstandard and nonprivileged port) that they or their friends can access from the Internet.

One common solution to the FTP problem is to use the SSH file transfer protocol. The protocol is currently an Internet draft but is widely used and mature. It is commonly used as a subcomponent of SSH, which provides its authentication and

7. In many cases, **inetd** or **xinetd** does the actual waiting on their behalf. See page 1188 for details.
8. Port 25 is the SMTP port as defined in **/etc/services**.
9. This summary describes traditional FTP, also known as "active FTP." Some systems support "passive FTP," in which the client initiates both connections.

encryption. Unlike FTP, SFTP uses only a single port for both commands and data, handily solving the packet-filtering paradox. A number of SFTP implementations exist. We've had great luck with the command-line SFTP client supplied by OpenSSH.

If you must use FTP, a reasonable approach is to allow FTP to the outside world only from a single, isolated host. Users can log in to the FTP machine when they need to perform network operations that are forbidden from the inner net. Since replicating all user accounts on the FTP "server" would defeat the goal of administrative separation, you may want to create FTP accounts by request only. Naturally, the FTP host should run a full complement of security-checking tools.

Modern security-conscious sites use a two-stage filtering scheme. In this scheme, one filter is a gateway to the Internet, and a second filter lies between the outer gateway and the rest of the local network. The idea is to terminate all inbound Internet connections on systems that lie in between these two filters. If these systems are administratively separate from the rest of the network, they can provide a variety of services to the Internet with reduced risk. The partially secured network is usually called the "demilitarized zone" or DMZ.

The most secure way to use a packet filter is to start with a configuration that allows no inbound connections. You can then liberalize the filter bit by bit as you discover useful things that don't work and, hopefully, move any Internet-accessible services onto systems in the DMZ.

Stateful inspection firewalls

The theory behind stateful inspection firewalls is that if you could carefully listen to and understand all the conversations (in all the languages) that were taking place in a crowded airport, you could make sure that someone wasn't planning to bomb a plane later that day. Stateful inspection firewalls are designed to inspect the traffic that flows through them and compare the actual network activity to what "should" be happening.

For example, if the packets exchanged in an FTP command sequence name a port to be used later for a data connection, the firewall should expect a data connection to occur only on that port. Attempts by the remote site to connect to other ports are presumably bogus and should be dropped.

So what are vendors really selling when they claim to provide stateful inspection? Their products either monitor a very limited number of connections or protocols or they search for a particular set of "bad" situations. Not that there's anything wrong with that; clearly, some benefit is derived from any technology that can detect traffic anomalies. In this particular case, however, it's important to remember that the claims are *mostly* marketing hype.

Firewalls: how safe are they?

A firewall should not be your primary (or only!) means of defense against intruders. It's only one component of what should be a carefully considered, multilayered security strategy. The use of firewalls often confers a false sense of security. If a firewall lulls you into relaxing other safeguards, it will have had a *negative* effect on the security of your site.

Every host within your organization should be individually patched, hardened, and regularly monitored with one or more tools such as Bro, Snort**,** Nmap, Nessus, and OSSEC. Likewise, your entire user community needs to be educated about basic security hygiene. Otherwise, you are simply building a structure that has a hard crunchy outside and a soft chewy center.

Ideally, local users should be able to connect to any Internet service they want, but machines on the Internet should only be able to connect to a limited set of local services hosted within your DMZ. For example, you may want to allow SFTP access to a local archive server and allow SMTP connections to a server that receives incoming email.

To maximize the value of your Internet connection, we recommend that you emphasize convenience and accessibility when deciding how to set up your network. At the end of the day, it's the system administrator's vigilance that makes a network secure, not a fancy piece of firewall hardware.

22.12 Linux firewall features

As stated earlier, we don't really recommended the use of Linux (or UNIX, or Windows) systems as firewalls because of the insecurity of running a full-fledged, general-purpose operating system.[10] However, a hardened Linux system is a workable substitute for organizations that don't have the budget for a high-dollar firewall appliance. Likewise, it's a fine option for a security-savvy home user with a penchant for tinkering. In any case, a local filter such as **iptables** can be an excellent supplemental security measure to consider when hardening a system.

If you are set on using a Linux machine as a firewall, make sure that it's up to date with respect to security configuration and patches. A firewall machine is an excellent place to put into practice all of this chapter's recommendations. (The section that starts on page 932 discusses packet-filtering firewalls in general. If you are not familiar with the basic concept of a firewall, it would probably be wise to read that section before continuing.)

Rules, chains, and tables

Version 2.4 of the Linux kernel introduced an all-new packet-handling engine, called Netfilter, along with a command-line tool, **iptables**, to manage it. **iptables**

10. That said, many consumer-oriented networking devices, such as Linksys's router products, use Linux and **iptables** at their core.

applies ordered "chains" of rules to network packets. Sets of chains make up "tables" and are used for handling specific kinds of traffic.

For example, the default **iptables** table is named "filter". Chains of rules in this table are used for packet-filtering network traffic. The filter table contains three default chains: FORWARD, INPUT, and OUTPUT. Each packet handled by the kernel is passed through exactly one of these chains.

Rules in the FORWARD chain are applied to all packets that arrive on one network interface and need to be forwarded to another. Rules in the INPUT and OUTPUT chains are applied to traffic addressed to or originating from the local host, respectively. These three standard chains are usually all you need for firewalling between two network interfaces. If necessary, you can define a custom configuration to support more complex accounting or routing scenarios.

In addition to the filter table, **iptables** includes the "nat" and "mangle" tables. The nat table contains chains of rules that control Network Address Translation (here, "nat" is the name of the **iptables** table and "NAT" is the name of the generic address translation scheme). The section *Private addresses and network address translation (NAT)* on page 462 discusses NAT, and an example of the nat table in action is shown on page 493. Later in this section, we use the nat table's PREROUTING chain for anti-spoofing packet filtering.

The mangle table contains chains that modify or alter the contents of network packets outside the context of NAT and packet filtering. Although the mangle table is handy for special packet handling, such as resetting IP time-to-live values, it is not typically used in most production environments. We discuss only the filter and nat tables in this section, leaving the mangle table to the adventurous.

Rule targets

Each rule that makes up a chain has a "target" clause that determines what to do with matching packets. When a packet matches a rule, its fate is in most cases sealed; no additional rules will be checked. Although many targets are defined internally to **iptables**, it is possible to specify another chain as a rule's target.

The targets available to rules in the filter table are ACCEPT, DROP, REJECT, LOG, MIRROR, QUEUE, REDIRECT, RETURN, and ULOG. When a rule results in an ACCEPT, matching packets are allowed to proceed on their way. DROP and REJECT both drop their packets; DROP is silent, and REJECT returns an ICMP error message. LOG gives you a simple way to track packets as they match rules, and ULOG expands logging.

See page 974 for more information about Squid.

REDIRECT shunts packets to a proxy instead of letting them go on their merry way. For example, you might use this feature to force all your site's web traffic to go through a web cache such as Squid. RETURN terminates user-defined chains and is analogous to the return statement in a subroutine call. The MIRROR target swaps the IP source and destination address before sending the packet. Finally, QUEUE hands packets to local user programs through a kernel module.

iptables firewall setup

Before you can use **iptables** as a firewall, you must enable IP forwarding and make sure that various **iptables** modules have been loaded into the kernel. For more information on enabling IP forwarding, see *Tuning Linux kernel parameters* on page 421 or *Security-related kernel variables* on page 492. Packages that install **iptables** generally include startup scripts to achieve this enabling and loading.

A Linux firewall is usually implemented as a series of **iptables** commands contained in an **rc** startup script. Individual **iptables** commands usually take one of the following forms:

```
iptables -F chain-name
iptables -P chain-name target
iptables -A chain-name -i interface -j target
```

The first form (**-F**) flushes all prior rules from the chain. The second form (**-P**) sets a default policy (aka target) for the chain. We recommend that you use DROP for the default chain target. The third form (**-A**) appends the current specification to the chain. Unless you specify a table with the **-t** argument, your commands apply to chains in the filter table. The **-i** parameter applies the rule to the named interface, and **-j** identifies the target. **iptables** accepts many other clauses, some of which are shown in Table 22.3.

Table 22.3 Command-line flags for iptables filters

Clause	Meaning or possible values
-p *proto*	Matches by protocol: **tcp**, **udp**, or **icmp**
-s *source-ip*	Matches host or network source IP address (CIDR notation is OK)
-d *dest-ip*	Matches host or network destination address
--sport *port#*	Matches by source port (note the double dashes)
--dport *port#*	Matches by destination port (note the double dashes)
--icmp-type *type*	Matches by ICMP type code (note the double dashes)
!	Negates a clause
-t *table*	Specifies the table to which a command applies (default is filter)

A complete example

Below we break apart a complete example. We assume that the eth1 interface goes to the Internet and that the eth0 interface goes to an internal network. The eth1 IP address is 128.138.101.4, the eth0 IP address is 10.1.1.1, and both interfaces have a netmask of 255.255.255.0. This example uses stateless packet filtering to protect the web server with IP address 10.1.1.2, which is the standard method of protecting Internet servers. Later in the example, we show how to use stateful filtering to protect desktop users.

Our first set of rules initializes the filter table. First, all chains in the table are flushed, then the INPUT and FORWARD chains' default target is set to DROP. As

with any other network firewall, the most secure strategy is to drop any packets you have not explicitly allowed.

```
iptables -F
iptables -P INPUT DROP
iptables -P FORWARD DROP
```

Since rules are evaluated in order, we put our busiest rules at the front.[11] The first rule allows all connections through the firewall that originate from within the trusted net. The next three rules in the FORWARD chain allow connections through the firewall to network services on 10.1.1.2. Specifically, we allow SSH (port 22), HTTP (port 80), and HTTPS (port 443) through to our web server.

```
iptables -A FORWARD -i eth0 -p ANY -j ACCEPT
iptables -A FORWARD -d 10.1.1.2 -p tcp --dport 22 -j ACCEPT
iptables -A FORWARD -d 10.1.1.2 -p tcp --dport 80 -j ACCEPT
iptables -A FORWARD -d 10.1.1.2 -p tcp --dport 443 -j ACCEPT
```

The only TCP traffic we allow to our firewall host (10.1.1.1) is SSH, which is useful for managing the firewall itself. The second rule listed below allows loopback traffic, which stays local to the host. Administrators get nervous when they can't **ping** their default route, so the third rule here allows ICMP ECHO_REQUEST packets from internal IP addresses.

```
iptables -A INPUT -i eth0 -d 10.1.1.1 -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -i lo -d 127.0.0.1 -p ANY -j ACCEPT
iptables -A INPUT -i eth0 -d 10.1.1.1 -p icmp --icmp-type 8 -j ACCEPT
```

For any IP host to work properly on the Internet, certain types of ICMP packets must be allowed through the firewall. The following eight rules allow a minimal set of ICMP packets to the firewall host, as well as to the network behind it.

```
iptables -A INPUT -p icmp --icmp-type 0 -j ACCEPT
iptables -A INPUT -p icmp --icmp-type 3 -j ACCEPT
iptables -A INPUT -p icmp --icmp-type 5 -j ACCEPT
iptables -A INPUT -p icmp --icmp-type 11 -j ACCEPT
iptables -A FORWARD -d 10.1.1.2 -p icmp --icmp-type 0 -j ACCEPT
iptables -A FORWARD -d 10.1.1.2 -p icmp --icmp-type 3 -j ACCEPT
iptables -A FORWARD -d 10.1.1.2 -p icmp --icmp-type 5 -j ACCEPT
iptables -A FORWARD -d 10.1.1.2 -p icmp --icmp-type 11 -j ACCEPT
```

See page 473 for more information about IP spoofing.

We next add rules to the PREROUTING chain in the nat table. Although the nat table is not intended for packet filtering, its PREROUTING chain is particularly useful for anti-spoofing filtering. If we put DROP entries in the PREROUTING chain, they need not be present in the INPUT and FORWARD chains, since the PREROUTING chain is applied to all packets that enter the firewall host. It's cleaner to put the entries in a single place rather than to duplicate them.

11. However, you must be careful that reordering the rules for performance doesn't modify functionality.

```
iptables -t nat -A PREROUTING -i eth1 -s 10.0.0.0/8 -j DROP
iptables -t nat -A PREROUTING -i eth1 -s 172.16.0.0/12 -j DROP
iptables -t nat -A PREROUTING -i eth1 -s 192.168.0.0/16 -j DROP
iptables -t nat -A PREROUTING -i eth1 -s 127.0.0.0/8 -j DROP
iptables -t nat -A PREROUTING -i eth1 -s 224.0.0.0/4 -j DROP
```

Finally, we end both the INPUT and FORWARD chains with a rule that forbids all packets not explicitly permitted. Although we already enforced this behavior with the **iptables -P** commands, the LOG target lets us see who is knocking on our door from the Internet.

```
iptables -A INPUT -i eth1 -j LOG
iptables -A FORWARD -i eth1 -j LOG
```

Optionally, we could set up IP NAT to disguise the private address space used on the internal network. See page 492 for more information about NAT.

One of the most powerful features that Netfilter brings to Linux firewalling is stateful packet filtering. Instead of allowing specific incoming services, a firewall for clients connecting to the Internet needs to allow incoming responses to the client's requests. The simple stateful FORWARD chain below allows all traffic to leave our network but only allows incoming traffic that's related to connections initiated by our hosts.

```
iptables -A FORWARD -i eth0 -p ANY -j ACCEPT
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Certain kernel modules must be loaded to enable **iptables** to track complex network sessions such as those of FTP and IRC. If these modules are not loaded, **iptables** simply disallows those connections. Although stateful packet filters can increase the security of your site, they also add to the complexity of the network. Be sure you need stateful functionality before implementing it in your firewall.

Perhaps the best way to debug your **iptables** rulesets is to use **iptables -L -v**. These options tell you how many times each rule in your chains has matched a packet. We often add temporary **iptables** rules with the LOG target when we want more information about the packets that get matched. You can often solve trickier problems by using a packet sniffer such as **tcpdump**.

22.13 IPFILTER FOR UNIX SYSTEMS

Most UNIX vendors don't have their own firewall software.[12] But it's easy enough to add: IPFilter, an open source package developed by Darren Reed, supplies NAT and stateful firewall services for UNIX systems. Solaris includes it by default, and it's also available as an add-on for HP-UX, AIX, and many other systems, including Linux. You can use IPFilter as a loadable kernel module (which is recommended by the developers) or include it statically in the kernel.

12. IBM is an exception. AIX does include a separate packet-filtering suite in its IP Security implementation, although the suite does not do stateful filtering. See the man pages for **genfilt** to get started.

IPFilter is mature and feature-complete. The package has an active user community and a history of continuous development. It is capable of stateful tracking even for stateless protocols such as UDP and ICMP.

IPFilter reads filtering rules from a configuration file (usually **/etc/ipf.conf** or **/etc/ipf/ipf.conf**) rather than making you run a series of commands as does **iptables**. An example of a simple rule that could appear in **ipf.conf** is

```
block in all
```

This rule blocks all inbound traffic (that is, network activity received by the system) on all network interfaces. Certainly secure, but not particularly useful!

Table 22.4 shows some of the possible conditions that can appear in an **ipf** rule.

Table 22.4 Commonly used ipf conditions

Condition	Meaning or possible values
on *interface*	Applies the rule to the specified interface
proto *protocol*	Selects packet according to protocol: tcp, udp, or icmp
from *source-ip*	Filters by source: host, network, or any
to *dest-ip*	Filters by destination: host, network, or any
port = *port#*	Filters by port name (from **/etc/services**) or number[a]
flags *flag-spec*	Filters according to TCP header flags bits
icmp-type *number*	Filters by ICMP type and code
keep state	Retains details about the flow of a session; see comments below

a. You can use any comparison operator: =, <, >, <=, >=, etc.

IPFilter evaluates rules in the sequence in which they are presented in the configuration file. The *last* match is binding. For example, inbound packets traversing the following filter will always pass:

```
block in all
pass in all
```

The block rule matches all packets, but so does the pass rule, and pass is the last match. To force a matching rule to apply immediately and make IPFilter skip subsequent rules, use the quick keyword:

```
block in quick all
pass in all
```

An industrial-strength firewall typically contains many rules, so liberal use of quick is important to maintain the performance of the firewall. Without it, every packet is evaluated against every rule, and this wastefulness is costly.

Perhaps the most common use of a firewall is to control access to and from a specific network or host, often with respect to a specific port. IPFilter has powerful syntax to control traffic at this level of granularity. In the following rules,

inbound traffic is permitted to the 10.0.0.0/24 network on TCP ports 80 and 443 and on UDP port 53.

```
block out quick all
pass in quick proto tcp from any to 10.0.0.0/24 port = 80 keep state
pass in quick proto tcp from any to 10.0.0.0/24 port = 443 keep state
pass in quick proto udp from any to 10.0.0.0/24 port = 53 keep state
block in all
```

The `keep state` keywords deserve special attention. IPFilter can keep track of connections by noting the first packet of new sessions. For example, when a new packet arrives addressed to port 80 on 10.0.0.10, IPFilter makes an entry in the state table and allows the packet through. It also allows the reply from the web server even though the first rule explicitly blocks all outbound traffic.

`keep state` is also useful for devices that offer no services but that must initiate connections. The following ruleset permits all conversations that are initiated by 192.168.10.10. It blocks all inbound packets except those related to connections that have already been initiated.

```
block in quick all
pass out quick from 192.168.10.10/32 to any keep state
```

The `keep state` keywords work for UDP and ICMP packets, too, but since these protocols are stateless, the mechanics are slightly more ad hoc: IPFilter permits responses to a UDP or an ICMP packet for 60 seconds after the inbound packet is seen by the filter. For example, if a UDP packet from 10.0.0.10, port 32,000, is addressed to 192.168.10.10, port 53, a UDP reply from 192.168.10.10 will be permitted until 60 seconds have passed. Similarly, an ICMP echo reply (ping response) is permitted after an echo request has been entered in the state table.

See page 462 for more information about private addresses and NAT.

Network address translation (NAT) is another feature offered by IPFilter. NAT lets a large network that uses RFC1918 private IP addresses connect to the Internet through a small set of Internet-routable IP addresses. The NAT device maps traffic from the private network to one or more public addresses, sends requests across the Internet, and then intercepts the responses and rewrites them in terms of the local IP addresses.

IPFilter uses the `map` keyword (in place of `pass` and `block`) to provide NAT services. In the following rule, traffic from the 10.0.0.0/24 network is mapped to the current routable address on the e1000g0 interface.

```
map e1000g0 10.0.0.0/24 -> 0/32
```

The filter must be reloaded if the address of e1000g0 changes, as might happen if e1000g0 leases a dynamic IP address through DHCP. For this reason, IPFilter's NAT features are best used at sites that have a static IP address on the Internet-facing interface.

IPFilter rules are flexible and configurable. Advanced features such as macros can considerably simplify the rules files. For details on these advanced features, see the official IPFilter site at coombs.anu.edu.au/~avalon.

The IPFilter package includes several commands, listed in Table 22.5.

Table 22.5 IPFilter commands

Cmd	Function
ipf	Manages rules and filter lists
ipfstat	Obtains statistics about packet filtering
ipmon	Monitors logged filter information
ipnat	Manages NAT rules

Of the commands in Table 22.5, **ipf** is the most commonly used. **ipf** accepts a rule file as input and adds correctly parsed rules to the kernel's filter list. **ipf** adds rules to the end of the filter unless you use the **-Fa** argument, which flushes all existing rules. For example, to flush the kernel's existing set of filters and load the rules from **ipf.conf**, use the following syntax:

```
solaris$ sudo ipf -Fa -f /etc/ipf/ipf.conf
```

IPFilter relies on pseudo-device files in **/dev** for access control, and by default only root can edit the filter list. We recommend leaving the default permissions in place and using **sudo** to maintain the filter.

Use **ipf**'s **-v** flag when loading the rules file to debug syntax errors and other problems in the configuration.

IPFilter is preinstalled in the Solaris kernel, but you must enable it with

```
solaris$ sudo svcadm enable network/ipfilter
```

before you can use it.

22.14 VIRTUAL PRIVATE NETWORKS (VPNS)

In its simplest form, a VPN is a connection that makes a remote network appear as if it is directly connected, even if it is physically thousands of miles and many router hops away. For increased security, the connection is not only authenticated in some way (usually with a "shared secret" such as a password), but the end-to-end traffic is also encrypted. Such an arrangement is usually referred to as a "secure tunnel."

Here's a good example of the kind of situation in which a VPN is handy: Suppose that a company has offices in Chicago, Boulder, and Miami. If each office has a connection to a local ISP, the company can use VPNs to transparently (and, for the most part, securely) connect the offices across the untrusted Internet. The

company could achieve a similar result by leasing dedicated lines to connect the three offices, but that would be considerably more expensive.

Another good example is a company whose employees telecommute from their homes. VPNs would allow those users to reap the benefits of their high-speed and inexpensive cable modem service while still making it appear that they are directly connected to the corporate network.

Because of the convenience and popularity of this functionality, everyone and his brother is offering some type of VPN solution. You can buy it from your router vendor as a plug-in for your operating system or even as a dedicated VPN device for your network. Depending on your budget and scalability needs, you may want to consider one of the many commercial VPN solutions.

If you're without a budget and looking for a quick fix, SSH can do secure tunneling for you. See the end of the SSH section on page 926.

IPsec tunnels

If you're a fan of IETF standards (or of saving money) and need a real VPN solution, take a look at IPsec (Internet Protocol security). IPsec was originally developed for IPv6, but it has also been widely implemented for IPv4. IPsec is an IETF-approved, end-to-end authentication and encryption system. Almost all serious VPN vendors ship a product that has at least an IPsec compatibility mode. Linux, Solaris, HP-UX, and AIX all include native kernel support for IPsec.

IPsec uses strong cryptography to provide both authentication and encryption services. Authentication ensures that packets are from the right sender and have not been altered in transit, and encryption prevents the unauthorized examination of packet contents.

In tunnel mode, IPsec encrypts the transport layer header, which includes the source and destination port numbers. Unfortunately, this scheme conflicts with the way in which most firewalls work. For this reason, most modern implementations default to using transport mode, in which only the payloads of packets (the data being transported) are encrypted.

There's a gotcha involving IPsec tunnels and MTU size. It's important to ensure that once a packet has been encrypted by IPsec, nothing fragments it along the path the tunnel traverses. To achieve this feat, you may have to lower the MTU on the devices in front of the tunnel (in the real world, 1,400 bytes usually works). See page 453 in the TCP chapter for more information about MTU size.

All I need is a VPN, right?

Sadly, there's a downside to VPNs. Although they do build a (mostly) secure tunnel across the untrusted network between the two endpoints, they don't usually address the security of the endpoints themselves. For example, if you set up a VPN between your corporate backbone and your CEO's home, you may be

inadvertently creating a path for your CEO's 15-year-old daughter to have direct access to everything on your network.

Bottom line: you need to treat connections from VPN tunnels as external connections and grant them additional privileges only as necessary and after careful consideration. Consider adding a special section to your site security policy that covers the rules that apply to VPN connections.

22.15 CERTIFICATIONS AND STANDARDS

If the subject matter of this chapter seems daunting to you, don't fret. Computer security is a complicated and vast topic, as countless books, web sites, and magazines can attest. Fortunately, much has been done to help quantify and organize the available information. Dozens of standards and certifications exist, and mindful system administrators should consider their guidance.

One of the most basic philosophical principles in information security is informally referred to as the "CIA triad."

The acronym stands for

- Confidentiality
- Integrity
- Availability

Confidentiality concerns the privacy of data. Access to information should be limited to those who are authorized to have it. Authentication, access control, and encryption are a few of the subcomponents of confidentiality. If a hacker breaks into a system and steals a database containing customer contact information, a compromise of confidentiality has occurred.

Integrity relates to the authenticity of information. Data integrity technology ensures that information is valid and has not been altered in any unauthorized way. It also addresses the trustworthiness of information sources. When a secure web site presents a signed SSL certificate, it is proving to the user not only that the information it is sending is encrypted but also that a trusted certificate authority (such as VeriSign or Equifax) has verified the identity of the source. Technologies such as PGP and Kerberos also guarantee data integrity.

Availability expresses the idea that information must be accessible to authorized users when they need it or there is no purpose in having it. Outages not caused by intruders, such as those caused by administrative errors or power outages, also fall into the category of availability problems. Unfortunately, availability is often ignored until something goes wrong.

Consider the CIA principles as you design, implement, and maintain systems. As the old security adage goes, "security is a process."

Certifications

This crash course in CIA is just a brief introduction to the larger information security field. Large corporations often employ many full-time employees whose job is guarding information. To gain credibility in the field and keep their knowledge current, these professionals attend training courses and obtain certifications. Prepare yourself for acronym-fu as we work through a few of the most popular certifications.

One of the most widely recognized security certifications is the CISSP, or Certified Information Systems Security Professional. It is administered by $(ISC)^2$, the International Information Systems Security Certification Consortium (say *that* ten times fast!). One of the primary draws of the CISSP is $(ISC)^2$'s notion of a "common body of knowledge" (CBK), essentially an industry-wide best practices guide for information security. The CBK covers law, cryptography, authentication, physical security, and much more. It's an incredible reference for security folks.

One criticism of the CISSP has been its concentration on breadth and consequent lack of depth. So many topics in the CBK, and so little time! To address this, $(ISC)^2$ has issued CISSP concentration programs that focus on architecture, engineering, and management. These specialized certifications add depth to the more general CISSP certification.

The System Administration, Networking, and Security (SANS) Institute created the Global Information Assurance Certification (GIAC) suite of certifications in 1999. Three dozen separate exams cover the realm of information security with tests divided into five categories. The certifications range in difficulty from the moderate two-exam GISF to the 23-hour, expert-level GSE. The GSE is notorious as one of the most difficult certifications in the industry. Many of the exams focus on technical specifics and require quite a bit of experience.

Finally, the Certified Information Systems Auditor (CISA) credential is an audit and process certification. It focuses on business continuity, procedures, monitoring, and other management content. Some consider the CISA an intermediate certification that is appropriate for an organization's security officer role. One of its most attractive aspects is that it involves only a single exam.

Although certifications are a personal endeavor, their application to business is undeniable. More and more companies now recognize certifications as the mark of an expert. Many businesses offer higher pay and promotions to certified employees. If you decide to pursue a certification, work closely with your organization to have it help pay for the associated costs.

Security standards

Because of the ever-increasing reliance on data systems, laws and regulations have been created to govern the management of sensitive, business-critical information. Major pieces of U.S. legislation such as HIPAA, FISMA, NERC CIP, and the

Sarbanes-Oxley Act (SOX) have all included sections on IT security. Although the requirements are sometimes expensive to implement, they have helped give the appropriate level of focus to a once-ignored aspect of technology.

For a broader discussions of industry and legal standards that affect IT environments, see page 1222.

Unfortunately, the regulations are filled with legalese and can be difficult to interpret. Most do not contain specifics on how to achieve their requirements. As a result, standards have been developed to help administrators reach the lofty legislative requirements. These standards are not regulation specific, but following them usually ensures compliance. It can be intimidating to confront the requirements of all the various standards at once, so it's usually best to first work through one standard in its entirety.

ISO 27002

The ISO/IEC 27002 (formerly ISO 17799) standard is probably the most widely accepted in the world. First introduced in 1995 as a British standard, it is 34 pages long and is divided into 11 sections that run the gamut from policy to physical security to access control. Objectives within each section define specific requirements, and controls under each objective describe the suggested "best practice" solutions. The document costs about $200.

The requirements are nontechnical and can be fulfilled by any organization in a way that best fits its needs. On the downside, the general wording of the standard leaves the reader with a sense of broad flexibility. Critics complain that the lack of specifics leaves organizations open to attack.

Nonetheless, this standard is one of the most valuable documents available to the information-security industry. It bridges an often tangible gap between management and engineering and helps focus both parties on minimizing risk.

PCI DSS

The Payment Card Industry Data Security Standard (PCI DSS) is a different beast entirely. It arose out of the perceived need to improve security in the credit card processing industry following a series of dramatic exposures. For example, in June 2005, CardSystems Services International revealed the "loss" of 40 million credit card numbers.

The U.S. Department of Homeland Security has estimated that $49.3 billion was lost to identity theft in 2009 alone. Not all of this can be linked directly to credit card exposure, of course, but increased vigilance by vendors would certainly have had a positive impact. The FBI has even connected credit card fraud to the funding of terrorist groups. Specific incidents include the bombings in Bali and the Madrid subway system.

The PCI DSS standard is the result of a joint effort between Visa and MasterCard, though it is currently maintained by Visa. Unlike ISO 27002, it is freely available for anyone to download. It focuses entirely on protecting cardholder data systems and has 12 sections that define requirements for protection.

Because PCI DSS is focused on card processors, it is not generally appropriate for businesses that don't deal with credit card data. However, for those that do, strict compliance is necessary to avoid hefty fines and possible criminal prosecution. You can find the document at pcisecuritystandards.org.

NIST 800 series

The fine folks at the National Institute of Standards and Technology (NIST) have created the Special Publication (SP) 800 series of documents to report on their research, guidelines, and outreach efforts in computer security. These documents are most often used in connection with measuring FISMA compliance for those organizations that handle data for the U.S. federal government. More generally, they are publicly available standards with excellent content and have been widely adopted by industry.

The SP 800 series includes more than 100 documents. All of them are available from csrc.nist.gov/publications/PubsSPs.html. Here are a few that you might want to consider starting with: NIST 800-12, *An Introduction to Computer Security: The NIST Handbook*; NIST 800-14, *Generally Accepted Principles and Practices for Securing Information Technology Systems*; NIST 800-34 R1, *Contingency Planning Guide for Information Technology Systems*; NIST 800-39, *Managing Risk from Information Systems: An Organizational Perspective*; NIST 800-53 R3, *Recommended Security Controls for Federal Information Systems and Organizations*; NIST 800-123, *Guide to General Server Security.*

Common Criteria

The Common Criteria for Information Technology Security Evaluation (commonly known as the "Common Criteria") is a standard against which to evaluate the security level of IT products. These guidelines have been established by an international committee of members from a variety of manufacturers and industries. See commoncriteriaportal.org to learn more about the standard and certified products.

OWASP

The Open Web Application Security Project (OWASP) is a not-for-profit worldwide organization focused on improving the security of application software. They are best known for their "top 10" list of web application security risks, which serves to remind all of us where to focus our energies when securing applications. Find the current list and a bunch of other great material at owasp.org.

22.16 SOURCES OF SECURITY INFORMATION

Half the battle of keeping your system secure consists of staying abreast of security-related developments in the world at large. If your site is broken into, the break-in probably won't be through the use of a novel technique. More likely, the

chink in your armor is a known vulnerability that has been widely discussed in vendor knowledge bases, on security-related newsgroups, and on mailing lists.

CERT: a registered service mark of Carnegie Mellon University

In response to the uproar over the 1988 Robert Morris, Jr., Internet worm, the Defense Advanced Research Projects Agency (DARPA) formed an organization called CERT, the Computer Emergency Response Team, to act as a clearing house for computer security information. CERT is still the best-known point of contact for security information, although it seems to have grown rather sluggish and bureaucratic of late. CERT also now insists that the name CERT does not stand for anything and is merely "a registered service mark of Carnegie Mellon University."

In mid-2003, CERT partnered with the Department of Homeland Security's National Cyber Security Division, NCSD. For better or worse, the merger has altered the previous mailing list structure.

The combined organization, known as US-CERT, offers four announcement lists, the most useful of which is the "Technical Cyber Security Alerts." Subscribe to any of the four lists at forms.us-cert.gov/maillists.

SecurityFocus.com and the BugTraq mailing list

SecurityFocus.com specializes in security-related news and information. The news includes current articles on general issues and on specific problems; there's also an extensive technical library of useful papers, nicely sorted by topic.

SecurityFocus's archive of security tools includes software for a variety of operating systems, along with blurbs and user ratings. It is the most comprehensive and detailed source of tools that we are aware of.

The BugTraq list is a moderated forum for the discussion of security vulnerabilities and their fixes. To subscribe, visit securityfocus.com/archive. Traffic on this list can be fairly heavy, however, and the signal-to-noise ratio is poor. A database of BugTraq vulnerability reports is also available from the web site.

Schneier on Security

Bruce Schneier's blog is a valuable and sometimes entertaining source of information about computer security and cryptography. Schneier is the author of the well-respected books *Applied Cryptography* and *Secrets and Lies*, among others. Information from the blog is also captured in the form of a monthly newsletter known as the Crypto-Gram. Learn more at schneier.com/crypto-gram.html.

SANS: the System Administration, Networking, and Security Institute

SANS is a professional organization that sponsors security-related conferences and training programs, as well as publishing a variety of security information. Their web site, sans.org, is a useful resource that occupies something of a middle

ground between SecurityFocus and CERT: neither as frenetic as the former nor as stodgy as the latter.

SANS offers several weekly and monthly email bulletins that you can sign up for on their web site. The weekly NewsBites are nourishing, but the monthly summaries seem to contain a lot of boilerplate. Neither is a great source of late-breaking security news.

Vendor-specific security resources

Because security problems have the potential to generate a lot of bad publicity, vendors are often eager to help customers keep their systems secure. Most large vendors have an official mailing list to which security-related bulletins are posted, and many maintain a web site about security issues as well. It's common for security-related software patches to be distributed for free, even by vendors that normally charge for software support.

Security portals on the web, such as SecurityFocus.com, contain vendor-specific information and links to the latest official vendor dogma.

Ubuntu maintains a security mailing list at

https://lists.ubuntu.com/mailman/listinfo/ubuntu-security-announce

You can find SUSE security advisories at

novell.com/linux/security/securitysupport.html

You can join the official SUSE security announcement mailing list by visiting

suse.com/en/private/support/online_help/mailinglists/index.html

Subscribe to the "Enterprise watch" list to get announcements about the security of Red Hat's product line at redhat.com/mailman/listinfo/enterprise-watch-list.

Despite Oracle's acquisition of Sun Microsystems, Sun's original security blog at blogs.sun.com/security/category/alerts continues to be updated. When the branding and location are updated, you can probably still find a pointer there.

You can access HP's various offerings through us-support.external.hp.com for the Americas and Asia, and europe-support.external.hp.com for Europe.

The security-related goodies have been carefully hidden. To find them, enter the maintenance/support area and select the option to search the technical knowledge base. A link in the filter options on that page takes you to the archive of security bulletins. (You will need to register if you have not already done so.) You can access security patches from that area as well.

To have security bulletins sent to you, return to the maintenance/support main page and choose the "Subscribe to proactive notifications and security bulletins" option. Unfortunately, there does not appear to be any way to subscribe directly by email.

Sign up for AIX security notifications through the "My notifications" link at ibm.com/systems/support.

Security information about Cisco products is distributed in the form of field notices, a list of which can be found at cisco.com/public/support/tac/fn_index.html along with a news aggregation feed. To subscribe to Cisco's security mailing list, email majordomo@cisco.com with the line "subscribe cust-security-announce" in the message body.

Other mailing lists and web sites

The contacts listed above are just a few of the many security resources available on the net. Given the volume of info that's now available and the rapidity with which resources come and go, we thought it would be most helpful to point you toward some meta-resources.

One good starting point is the linuxsecurity.com, which logs several posts every day on pertinent Linux security issues. It also keeps a running collection of Linux security advisories, upcoming events, and user groups.

(IN)SECURE magazine is a free bimonthly magazine containing current security trends, product announcements, and interviews with notable security professionals. Read some of the articles with a vial of salt nearby, and be sure to check the author at the end—he may be pimping his own products.

Linux Security (linuxsecurity.com) covers the latest news in Linux and open source security. Subscribe to the RSS feed for best results.

The Linux Weekly News is a tasty treat that includes regular updates on the kernel, security, distributions, and other topics. LWN's security section can be found at lwn.net/security.

22.17 WHAT TO DO WHEN YOUR SITE HAS BEEN ATTACKED

The key to handling an attack is simple: don't panic. It's very likely that by the time you discover the intrusion, most of the damage has already been done. In fact, it has probably been going on for weeks or months. The chance that you've discovered a break-in that just happened an hour ago is slim to none.

In that light, the wise owl says to take a deep breath and begin developing a carefully thought out strategy for dealing with the break-in. You need to avoid tipping off the intruder by announcing the break-in or performing any other activity that would seem abnormal to someone who may have been watching your site's operations for many weeks. Hint: performing a system backup is usually a good idea at this point and (hopefully!) will appear to be a normal activity to the intruder.[13]

13. If system backups are not a "normal" activity at your site, you have much bigger problems than the security intrusion.

This is also a good time to remind yourself that some studies have shown that 60% of security incidents involve an insider. Be very careful with whom you discuss the incident until you're sure you have all the facts.

Here's a quick 9-step plan that may assist you in your time of crisis:

Step 1: Don't panic. In many cases, a problem isn't noticed until hours or days after it took place. Another few hours or days won't affect the outcome. The difference between a panicky response and a rational response will. Many recovery situations are exacerbated by the destruction of important log, state, and tracking information during an initial panic.

Step 2: Decide on an appropriate level of response. No one benefits from an over-hyped security incident. Proceed calmly. Identify the staff and resources that must participate and leave others to assist with the post-mortem after it's all over.

Step 3: Hoard all available tracking information. Check accounting files and logs. Try to determine where the original breach occurred. Back up all your systems. Make sure that you physically write-protect backup tapes if you put them in a drive to read them.

Step 4: Assess your degree of exposure. Determine what crucial information (if any) has "left" the company, and devise an appropriate mitigation strategy. Determine the level of future risk.

Step 5: Pull the plug. If necessary and appropriate, disconnect compromised machines from the network. Close known holes and stop the bleeding. CERT provides steps for analyzing an intrusion. The document can be found at

cert.org/tech_tips/win-UNIX-system_compromise.html

Step 6: Devise a recovery plan. With a creative colleague, draw up a recovery plan on nearby whiteboard. This procedure is most effective when performed away from a keyboard. Focus on putting out the fire and minimizing the damage. Avoid assessing blame or creating excitement. In your plan, don't forget to address the psychological fallout your user community may experience. Users inherently trust others, and blatant violations of trust make many folks uneasy.

Step 7: Communicate the recovery plan. Educate users and management about the effects of the break-in, the potential for future problems, and your preliminary recovery strategy. Be open and honest. Security incidents are part of life in a modern networked environment. They are not a reflection on your ability as a system administrator or on anything else worth being embarrassed about. Openly admitting that you have a problem is 90% of the battle, as long as you can demonstrate that you have a plan to remedy the situation.

Step 8: Implement the recovery plan. You know your systems and networks better than anyone. Follow your plan and your instincts. Speak with a colleague at a similar institution (preferably one who knows you well) to keep yourself on the right track.

Step 9: Report the incident to authorities. If the incident involved outside parties, report the matter to CERT. They have a hotline at (412) 268-7090 and can be reached by email at cert@cert.org. Provide as much information as you can.

A standard form is available from cert.org to help jog your memory. Here are some of the more useful pieces of information you might provide:

- The names, hardware, and OS versions of the compromised machines
- The list of patches that had been applied at the time of the incident
- A list of accounts that are known to have been compromised
- The names and IP addresses of any remote hosts that were involved
- Contact information (if known) for the administrators of remote sites
- Relevant log entries or audit information

If you believe that a previously undocumented software problem may have been involved, you should report the incident to the software vendor as well.

22.18 RECOMMENDED READING

BARRETT, DANIEL J., RICHARD E. SILVERMAN, AND ROBERT G. BYRNES. *Linux Security Cookbook.* Sebastopol, CA: O'Reilly Media, 2003.

BAUER, MICHAEL D. *Linux Server Security (2nd Edition).* Sebastopol, CA: O'Reilly Media, 2005.

BRYANT, WILLIAM. "Designing an Authentication System: a Dialogue in Four Scenes." 1988. web.mit.edu/kerberos/www/dialogue.html

CHESWICK, WILLIAM R., STEVEN M. BELLOVIN, AND AVIEL D RUBIN. *Firewalls and Internet Security: Repelling the Wily Hacker (2nd Edition).* Reading, MA: Addison-Wesley, 2003.

CURTIN, MATT, MARCUS RANUM, AND PAUL D. ROBINSON. "Internet Firewalls: Frequently Asked Questions." 2004. interhack.net/pubs/fwfaq

FARROW, RIK, AND RICHARD POWER. *Network Defense article series*. 1998-2004. spirit.com/Network

FRASER, B., EDITOR. *RFC2196: Site Security Handbook.* 1997. rfc-editor.org

GARFINKEL, SIMSON, GENE SPAFFORD, AND ALAN SCHWARTZ. *Practical UNIX and Internet Security (3rd Edition)*. Sebastopol, CA: O'Reilly Media, 2003.

KERBY, FRED, ET AL. "SANS Intrusion Detection and Response FAQ." SANS. 2009. sans.org/resources/idfaq

LYON, GORDON FYODOR. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Nmap Project, 2009.

MANN, SCOTT, AND ELLEN L. MITCHELL. *Linux System Security: The Administrator's Guide to Open Source Security Tools (2nd Edition)*. Upper Saddle River, NJ: Prentice Hall PTR, 2002.

MORRIS, ROBERT, AND KEN THOMPSON. "Password Security: A Case History." *Communications of the ACM*, 22 (11): 594-597, November 1979. Reprinted in *UNIX System Manager's Manual*, 4.3 Berkeley Software Distribution. University of California, Berkeley, April 1986.

RITCHIE, DENNIS M. "On the Security of UNIX." May 1975. Reprinted in *UNIX System Manager's Manual*, 4.3 Berkeley Software Distribution. University of California, Berkeley, April 1986.

SCHNEIER, BRUCE. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. New York, NY: Wiley, 1995.

STEVES, KEVIN. "Building a Bastion Host Using HP-UX 11." HP Consulting. 2002. tinyurl.com/5sffy2

THOMPSON, KEN. "Reflections on Trusting Trust." in *ACM Turing Award Lectures: The First Twenty Years 1966-1985*. Reading, MA: ACM Press (Addison-Wesley), 1987.

Exercises begin on the next page.

29 Performance Analysis

Performance analysis and tuning are often likened to system administration witchcraft. They're not really witchcraft, but they do qualify as both science and art. The "science" part involves making careful quantitative measurements and applying the scientific method. The "art" part relates to the need to balance resources in a practical, level-headed way, since optimizing for one application or user may result in other applications or users suffering. As with so many things in life, you may find that it's impossible to make everyone happy.

A sentiment widespread in the blogosphere has it that today's performance problems are somehow wildly different from those of previous decades. That claim is inaccurate. It's true that systems have become more complex, but the baseline determinants of performance and the high-level abstractions used to measure and manage it remain the same as always. Unfortunately, improvements in baseline system performance correlate strongly with the community's ability to create new applications that suck up all available resources.

This chapter focuses on the performance of systems that are used as servers. Desktop systems typically do not experience the same types of performance issues that servers do, and the answer to the question of how to improve performance on

a desktop machine is almost always "Upgrade the hardware." Users like this answer because it means they get fancy new systems on their desks more often.

One of the ways in which UNIX and Linux differ from other mainstream operating systems is in the amount of data that is available to characterize their inner workings. Detailed information is available for every level of the system, and administrators control a variety of tunable parameters. If you still have trouble identifying the cause of a performance problem despite the available instrumentation, source code is often available for review. For these reasons, UNIX and Linux are typically the operating systems of choice for performance-conscious consumers.

Even so, performance tuning isn't easy. Users and administrators alike often think that if they only knew the right "magic," their systems would be twice as fast. One common fantasy involves tweaking the kernel variables that control the paging system and the buffer pools. These days, kernels are pretuned to achieve reasonable (though admittedly, not optimal) performance under a variety of load conditions. If you try to optimize the system on the basis of one particular measure of performance (e.g., buffer utilization), the chances are high that you will distort the system's behavior relative to other performance metrics and load conditions.

The most serious performance issues often lie within applications and have little to do with the underlying operating system. This chapter discusses system-level performance tuning and mostly leaves application-level tuning to others. As a system administrator, you need to be mindful that application developers are people too. (How many times have you said, or thought, that "it must be a network problem"?) Given the complexity of modern applications, some problems can only be resolved through collaboration among application developers, system administrators, server engineers, DBAs, storage administrators, and network architects. In this chapter, we help you determine what data and information to take back to these other folks to help them solve a performance problem—if, indeed, the problem lies in their area. This approach is far more productive than just saying, "Everything looks fine; it's not my problem."

In all cases, take everything you read on the web with a ~~tablespoon~~ cup of salt. In the area of system performance, you will see superficially convincing arguments on all sorts of topics. However, most of the proponents of these theories do not have the knowledge, discipline, and time required to design valid experiments. Popular support means very little; for every hare-brained proposal, you can expect to see a Greek chorus of "I increased the size of my buffer cache by a factor of ten just like Joe said, and my system feels *much, much* faster!!!" Right.

Here are some rules to keep in mind:

- Collect and review *historical* information about your system. If the system was performing fine a week ago, an examination of the aspects of the system that have changed may lead you to a smoking gun. Keep baselines and trends in your hip pocket to pull out in an emergency. As a first step, review log files to see if a hardware problem has developed.

Chapter 21, *Network Management and Debugging*, discusses some trend analysis tools that are also applicable to performance monitoring. The **sar** utility discussed on page 1129 can also be used as a poor man's trend analysis tool.

- Tune your system in a way that lets you compare the current results to the system's previous baseline.
- Always make sure you have a rollback plan in case your magic fix actually makes things worse.
- Don't intentionally overload your systems or your network. The kernel gives each process the illusion of infinite resources. But once 100% of the system's resources are in use, the kernel has to work hard to maintain that illusion, delaying processes and often consuming a sizable fraction of the resources itself.
- As in particle physics, the more information you collect with system monitoring utilities, the more you affect the system you are observing. It is best to rely on something simple and lightweight that runs in the background (e.g., **sar** or **vmstat**) for routine observation. If those feelers show something significant, you can investigate further with other tools.

29.1 WHAT YOU CAN DO TO IMPROVE PERFORMANCE

Here are some specific things you can do to improve performance:

- Ensure that the system has enough memory. As we see in the next section, memory size has a major influence on performance. Memory is so inexpensive these days that you can usually afford to load every performance-sensitive machine to the gills.
- If you are using UNIX or Linux as a web server or as some other type of network application server, you may want to spread traffic among several systems with a commercial load balancing appliance such as Cisco's Content Services Switch (cisco.com), Nortel's Alteon Application Switch (nortel.com), or Brocade's ServerIron (brocade.com). These boxes make several physical servers appear to be one logical server to the outside world. They balance the load according to one of several user-selectable algorithms such as "most responsive server" or "round robin."

  These load balancers also provide useful redundancy should a server go down. They're really quite necessary if your site must handle unexpected traffic spikes.
- Double-check the configuration of the system and of individual applications. Many applications can be tuned to yield tremendous performance improvements (e.g., by spreading data across disks, by not performing DNS lookups on the fly, or by running multiple instances of a server).

- Correct problems of usage, both those caused by "real work" (too many servers run at once, inefficient programming practices, batch jobs run at excessive priority, and large jobs run at inappropriate times of day) and those caused by the system (such as unwanted daemons).
- Eliminate storage resources' dependence on mechanical operations where possible. Solid state disk drives (SSDs) are widely available and can provide quick performance boosts because they don't require the physical movement of a disk or armature to read bits. SSDs are easily installed in place of existing old-school disk drives.[1]
- Organize hard disks and filesystems so that load is evenly balanced, maximizing I/O throughput. For specific applications such as databases, you can use a fancy multidisk technology such as striped RAID to optimize data transfers. Consult your database vendor for recommendations. For Linux systems, ensure that you've selected the appropriate Linux I/O scheduler for your disk (see page 1130 for details).
- It's important to note that different types of applications and databases respond differently to being spread across multiple disks. RAID comes in many forms; take time to determine which form (if any) is appropriate for your particular application.
- Monitor your network to be sure that it is not saturated with traffic and that the error rate is low. A wealth of network information is available through the **netstat** command, described on page 868. See also Chapter 21, *Network Management and Debugging*.
- Identify situations in which the system is fundamentally inadequate to satisfy the demands being made of it. You cannot tune your way out of these situations.

These steps are listed in rough order of effectiveness. Adding memory and balancing traffic across multiple servers can often make a huge difference in performance. The effectiveness of the other measures ranges from noticeable to none.

Analysis and optimization of software data structures and algorithms almost always lead to significant performance gains. But unless you have a substantial base of local software, this level of design is usually out of your control.

29.2 FACTORS THAT AFFECT PERFORMANCE

Perceived performance is determined by the basic capabilities of the system's resources and by the efficiency with which those resources are allocated and shared.

1. Current SSDs have two main weaknesses. First, they are an order of magnitude more expensive per gigabyte than traditional hard disks. Second, they may be rewritten only a limited number of times before wearing out. Their rewrite capacity is high enough to be immaterial for desktop machines (tens of thousands of writes per block), but it's a potential stumbling block for a high-traffic server. See page 212 for more information about SSDs.

The exact definition of a "resource" is rather vague. It can include such items as cached contexts on the CPU chip and entries in the address table of the memory controller. However, to a first approximation, only the following four resources have much effect on performance:

- CPU utilization
- Memory
- Storage I/O
- Network I/O

If resources are still left after active processes have taken what they want, the system's performance is about as good as it can be.

If there are not enough resources to go around, processes must take turns. A process that does not have immediate access to the resources it needs must wait around doing nothing. The amount of time spent waiting is one of the basic measures of performance degradation.

CPU utilization is one of the easiest resources to measure. A constant amount of processing power is always available. In theory, that amount is 100% of the CPU cycles, but overhead and various inefficiencies make the real-life number more like 95%. A process that's using more than 90% of the CPU is entirely CPU bound and is consuming essentially all of the system's available computing power.

Many people assume that the speed of the CPU is the most important factor affecting a system's overall performance. Given infinite amounts of all other resources or certain types of applications (e.g., numerical simulations), a faster CPU *does* make a dramatic difference. But in the everyday world, CPU speed is relatively unimportant.

Disk bandwidth is a common performance bottleneck. Because traditional hard disks are mechanical systems, it takes many milliseconds to locate a disk block, fetch its contents, and wake up the process that's waiting for it. Delays of this magnitude overshadow every other source of performance degradation. Each disk access causes a stall worth millions of CPU instructions. Solid state drives are one tool you can use to address this problem; they are significantly faster than drives with moving parts.

Because of virtual memory, disk bandwidth and memory can be directly related if the demand for physical memory is greater than the supply. Situations in which physical memory becomes scarce often result in memory pages being written to disk so they can be reclaimed and reused for another purpose. In these situations, using memory is just as expensive as using the disk. Avoid this trap when performance is important; ensure that every system has adequate physical memory.

Network bandwidth resembles disk bandwidth in many ways because of the latencies involved in network communication. However, networks are atypical in that they involve entire communities rather than individual computers. They are also particularly susceptible to hardware problems and overloaded servers.

29.3 HOW TO ANALYZE PERFORMANCE PROBLEMS

It can be difficult to isolate performance problems in a complex system. As a sysadmin, you often receive anecdotal problem reports that suggest a particular cause or fix (e.g., "The web server has gotten painfully sluggish because of all those damn AJAX calls…"). Take note of this information, but don't assume that it's accurate or reliable; do your own investigation.

A rigorous, transparent, scientific methodology helps you reach conclusions that you and others in your organization can rely on. Such an approach lets others evaluate your results, increases your credibility, and raises the likelihood that your suggested changes will actually fix the problem.

"Being scientific" doesn't mean that you have to gather all the relevant data yourself. External information is usually very helpful. Don't spend hours looking into issues that can just as easily be looked up in a FAQ.

We suggest the following five steps:

Step 1: Formulate the question.

Pose a specific question in a defined functional area, or state a tentative conclusion or recommendation that you are considering. Be specific about the type of technology, the components involved, the alternatives you are considering, and the outcomes of interest.

Step 2: Gather and classify evidence.

Conduct a systematic search of documentation, knowledge bases, known issues, blogs, white papers, discussions, and other resources to locate external evidence related to your question. On your own systems, capture telemetry data and, where necessary or possible, instrument specific system and application areas of interest.

Step 3: Critically appraise the data.

Review each data source for relevance and critique it for validity. Abstract key information and note the quality of the sources.

Step 4: Summarize the evidence both narratively and graphically.

Combine findings from multiple sources into a narrative précis and, if possible, a graphic representation. Data that seems equivocal in numeric form can become decisive once charted.

Step 5: Develop a conclusion statement.

Arrive at a concise statement of your conclusions (i.e., the answer to your question). Assign a grade to indicate the overall strength or weakness of the evidence that supports your conclusions.

29.4 SYSTEM PERFORMANCE CHECKUP

Enough generalities—let's look at some specific tools and areas of interest. Before you take measurements, you need to know what you're looking at.

Taking stock of your hardware

Start your inquiry with an inventory of your hardware, especially CPU and memory resources. This inventory can help you interpret the information presented by other tools and can help you set realistic expectations regarding the upper bounds on performance.

On Linux systems, the **/proc** filesystem is the place to look to find an overview of what hardware your operating system thinks you have (more detailed hardware information can be found in **/sys**; see page 438). Table 29.1 shows some of the key files. See page 421 for general information about **/proc**.

Table 29.1 Sources of hardware information on Linux

File	Contents
/proc/cpuinfo	CPU type and description
/proc/meminfo	Memory size and usage
/proc/diskstats	Disk devices and usage statistics

Four lines in **/proc/cpuinfo** help you identify the system's exact CPU: vendor_id, cpu family, model, and model name. Some of the values are cryptic; it's best to look them up on-line.

The exact info contained in **/proc/cpuinfo** varies by system and processor, but here's a representative example:

```
suse$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 15
model name      : Intel(R) Xeon(R) CPUE5310@ 1.60GHz
stepping        : 11
cpu MHz         : 1600.003
cache size      : 4096 KB
physical id     : 0
cpu cores       : 2
siblings        : 2
…
```

The file contains one entry for each processor core seen by the OS. The data varies slightly by kernel version. The processor value uniquely identifies each core. physical id values are unique per physical socket on the circuit board, and core id

values are unique per core within a physical socket. Cores that support hyperthreading (duplication of CPU contexts without duplication of other processing features) are identified by an ht in the flags field. If hyperthreading is actually in use, the siblings field for each core shows how many contexts are available on a given core.

Another command to run for information on PC hardware is **dmidecode**. It dumps the system's Desktop Management Interface (DMI, aka SMBIOS) data. The most useful option is **-t** *type*; Table 29.2 shows the valid *type*s.

Table 29.2 Type values for dmidecode -t

Value	Description
1	System information
2	Base board Information
3	Chassis information
4	Processor information
7	Cache information
8	Port connector information
9	System slot information
11	OEM strings
12	System configuration options
13	BIOS language information
16	Physical memory array
17	Memory device
19	Memory array mapped address
32	System boot information
38	IPMI device information

The example below shows typical information:

```
suse$ sudo dmidecode -t 4
# dmidecode 2.7
SMBIOS 2.2 present.

Handle 0x0004, DMI type 4, 32 bytes.
Processor Information
        Socket Designation: PGA 370
        Type: Central Processor
        Family: Celeron
        Manufacturer: GenuineIntel
        ID: 65 06 00 00 FF F9 83 01
        Signature: Type 0, Family 6, Model 6, Stepping 5
...
```

Bits of network configuration information are scattered about the system. **ifconfig -a** is the best source of IP and MAC information for each configured interface.

On Solaris systems, the **psrinfo -v** and **prtconf** commands are the best sources of information about CPU and memory resources, respectively. Example output for these commands is shown below.

```
solaris$ psrinfo -v
Status of virtual processor 0 as of: 01/31/2010 21:22:00
  on-line since 07/13/2009 15:55:48.
  The sparcv9 processor operates at 1200 MHz,
        and has a sparcv9 floating point processor.
Status of virtual processor 1 as of: 01/31/2010 21:22:00
  on-line since 07/13/2009 15:55:49.
  The sparcv9 processor operates at 1200 MHz,
        and has a sparcv9 floating point processor.
```

```
solaris$ prtconf
System Configuration:  Sun Microsystems  sun4v
Memory size: 32640 Megabytes
System Peripherals (Software Nodes):

SUNW,Sun-Fire-T200
…
```

Under HP-UX, **machinfo** is an all-in-one command you can use to investigate a machine's hardware configuration. Here's some typical output:

```
hp-ux$ sudo machinfo
CPU info:
  1 Intel(R) Itanium 2 processor (1.5 GHz, 6 MB)
        400 MT/s bus, CPU version B1

Memory: 4084 MB (3.99 GB)

Firmware info:
  Firmware revision:  02.21
  FP SWA driver revision: 1.18
  BMC firmware revision: 1.50

Platform info:
  Model:           "ia64 hp server rx2600"
OS info:
  Nodename:   hpux11
  Release:    HP-UX B.11.31
Machine:      ia64
```

It takes a bit of work to find CPU and memory information under AIX. First, use the **lscfg** command to find the names of the installed processors.

```
aix$ lscfg | grep Processor
+ proc0                                      Processor
+ proc2                                      Processor
```

You can then use **lsattr** to extract a description of each processor:

```
aix$ lsattr -E -l proc0
frequency     1898100000          Processor Speed          False
smt_enabled   true                Processor SMT enabled    False
smt_threads   2                   Processor SMT threads    False
state         enable              Processor state          False
type          PowerPC_POWER5      Processor type           False
```

lsattr can also tell you the amount of physical memory in the system:

```
aix$ lsattr -E -l sys0 -a realmem
realmem    4014080      Amount of usable physical memory in Kbytes
```

Gathering performance data

Most performance analysis tools tell you what's going on at a particular point in time. However, the number and character of loads probably changes throughout the day. Be sure to gather a cross-section of data before taking action. The best information on system performance often becomes clear only after a long period (a month or more) of data collection. It is particularly important to collect data during periods of peak use. Resource limitations and system misconfigurations are often only visible when the machine is under heavy load.

Analyzing CPU usage

You will probably want to gather three kinds of CPU data: overall utilization, load averages, and per-process CPU consumption. Overall utilization can help identify systems on which the CPU's speed is itself the bottleneck. Load averages give you an impression of overall system performance. Per-process CPU consumption data can identify specific processes that are hogging resources.

You can obtain summary information with the **vmstat** command. **vmstat** takes two arguments: the number of seconds to monitor the system for each line of output and the number of reports to provide. If you don't specify the number of reports, **vmstat** runs until you press <Control-C>.

The first line of data returned by **vmstat** reports averages since the system was booted. The subsequent lines are averages within the previous sample period, which defaults to five seconds. For example:

```
$ vmstat 5 5
procs      -----------memory----------  ---swap-------io----  --system--  ----cpu----
 r  b  swpd       free  buff       cache  si  so     bi  bo     in     cs  us  sy  id  wa
 1  0   820    2606356  428776    487092   0   0   4741  65   1063   4857  25   1  73   0
 1  0   820    2570324  428812    510196   0   0   4613  11   1054   4732  25   1  74   0
 1  0   820    2539028  428852    535636   0   0   5099  13   1057   5219  90   1   9   0
 1  0   820    2472340  428920    581588   0   0   4536  10   1056   4686  87   3  10   0
 3  0   820    2440276  428960    605728   0   0   4818  21   1060   4943  20   3  77   0
```

Although exact columns may vary among systems, CPU utilization stats are fairly consistent across platforms. User time, system (kernel) time, idle time, and time

waiting for I/O are shown in the us, sy, id, and wa columns on the far right. CPU numbers that are heavy on user time generally indicate computation, and high system numbers indicate that processes are making a lot of system calls or are performing lots of I/O.

A rule of thumb for general-purpose compute servers that has served us well over the years is that the system should spend approximately 50% of its nonidle time in user space and 50% in system space; the overall idle percentage should be nonzero. If you are dedicating a server to a single CPU-intensive application, the majority of time should be spent in user space.

The cs column shows context switches per interval (that is, the number of times that the kernel changed which process was running). The number of interrupts per interval (usually generated by hardware devices or components of the kernel) is shown in the in column. Extremely high cs or in values typically indicate a misbehaving or misconfigured hardware device. The other columns are useful for memory and disk analysis, which we discuss later in this chapter.

Long-term averages of the CPU statistics let you determine whether there is fundamentally enough CPU power to go around. If the CPU usually spends part of its time in the idle state, there are cycles to spare. Upgrading to a faster CPU won't do much to improve the overall throughput of the system, though it may speed up individual operations.

As you can see from this example, the CPU generally flip-flops back and forth between heavy use and idleness. Therefore, it's important to observe these numbers as an average over time. The smaller the monitoring interval, the less consistent the results.

On multiprocessor machines, most tools present an average of processor statistics across all processors. On Linux, Solaris, and AIX, the **mpstat** command generates **vmstat**-like output for each individual processor. The **-P** flag lets you specify a specific processor to report on. **mpstat** is useful for debugging software that supports symmetric multiprocessing—it's also enlightening to see how (in)efficiently your system uses multiple processors. Here's an example that shows the status of each of four processors:

```
linux$ mpstat -P ALL
08:13:38 PM  CPU  %user  %nice  %sys  %iowait  %irq  %soft  %idle   intr/s
08:13:38 PM    0   1.02   0.00  0.49     1.29  0.04   0.38  96.79   473.93
08:13:38 PM    1   0.28   0.00  0.22     0.71  0.00   0.01  98.76   232.86
08:13:38 PM    2   0.42   0.00  0.36     1.32  0.00   0.05  97.84   293.85
08:13:38 PM    3   0.38   0.00  0.30     0.94  0.01   0.05  98.32   295.02
```

On a workstation with only one user, the CPU generally spends most of its time idle. Then when you render a web page or switch windows, the CPU is used heavily for a short period. In this situation, information about long-term average CPU usage is not meaningful.

The second CPU statistic that's useful for characterizing the burden on your system is the "load average," which represents the average number of runnable processes. It gives you a good idea of how many pieces the CPU pie is being divided into. The load average is obtained with the **uptime** command:

```
$ uptime
11:10am  up 34 days, 18:42, 5 users, load average: 0.95, 0.38, 0.31
```

Three values are given, corresponding to the 5, 10, and 15-minute averages. In general, the higher the load average, the more important the system's aggregate performance becomes. If there is only one runnable process, that process is usually bound by a single resource (commonly disk bandwidth or CPU). The peak demand for that one resource becomes the determining factor in performance.

When more processes share the system, loads may or may not be more evenly distributed. If the processes on the system all consume a mixture of CPU, disk, and memory, the performance of the system is less likely to be dominated by constraints on a single resource. In this situation, it becomes most important to look at average measures of consumption, such as total CPU utilization.

See page 123 for more information about priorities.

Modern single-processor systems are typically busy with a load average of 3 and do not deal well with load averages over about 8. A load average of this magnitude is a hint that you should start to look for ways to spread the load artificially, such as by using **nice** to set process priorities.

The system load average is an excellent metric to track as part of a system baseline. If you know your system's load average on a normal day and it is in that same range on a bad day, this is a hint that you should look elsewhere (such as the network) for performance problems. A load average above the expected norm suggests that you should look at the processes running on the system itself.

Another way to view CPU usage is to run the **ps** command with arguments that show you how much of the CPU each process is using (**-aux** for Linux and AIX, **-elf** for HP-UX and Solaris). On a busy system, at least 70% of the CPU is often consumed by just one or two processes. Deferring the execution of the CPU hogs or reducing their priority makes the CPU more available to other processes.

*See page 133 for more information about **top**.*

An excellent alternative to **ps** is a program called **top**. It presents about the same information as **ps**, but in a live, regularly updated format that shows the status of the system over time.[2] AIX's **topas** command is even nicer.

On virtualized systems, **ps**, **top**, and other commands that display CPU utilization data may be misleading. A virtual machine that is not using all of its virtual CPU cycles allows other virtual machines to use (steal) those cycles. Any measurement that is relative to the operating system, such as clock ticks per second, should be examined carefully to be sure you understand what is really being reported. See

2. Refreshing **top**'s output too rapidly can itself be quite a CPU hog, so be judicious in your use of **top**.

Chapter 24, *Virtualization*, for additional information about various virtualization technologies and their implications.

How the system manages memory

The kernel manages memory in units called pages that are usually 4KiB or larger. It allocates virtual pages to processes as they request memory. Each virtual page is mapped to real storage, either to RAM or to "backing store" on disk. (Backing store is usually space in the swap area, but for pages that contain executable program text, the backing store is the original executable file. Likewise, the backing store for some data files may be the files themselves.) The kernel uses a "page table" to keep track of the mappings between these made-up virtual pages and real pages of memory.

The kernel can effectively allocate as much memory as processes ask for by augmenting real RAM with swap space. Since processes expect their virtual pages to map to real memory, the kernel may have to constantly shuffle pages between RAM and swap as different pages are accessed. This activity is known as paging.[3]

The kernel tries to manage the system's memory so that pages that have been recently accessed are kept in memory and less active pages are paged out to disk. This scheme is known as an LRU system since the least recently used pages are the ones that get shunted to disk.

It would be inefficient for the kernel to keep track of all memory references, so it uses a cache-like algorithm to decide which pages to keep in memory. The exact algorithm varies by system, but the concept is similar across platforms. This system is cheaper than a true LRU system and produces comparable results.

When memory is low, the kernel tries to guess which pages on the inactive list were least recently used. If those pages have been modified by a process, they are considered "dirty" and must be paged out to disk before the memory can be reused. Pages that have been laundered in this fashion (or that were never dirty to begin with) are "clean" and can be recycled for use elsewhere.

When a process refers to a page on the inactive list, the kernel returns the page's memory mapping to the page table, resets the page's age, and transfers it from the inactive list to the active list. Pages that have been written to disk must be paged in before they can be reactivated if the page in memory has been remapped. A "soft fault" occurs when a process references an in-memory inactive page, and a "hard fault" results from a reference to a nonresident (paged-out) page. In other words, a hard fault requires a page to be read from disk and a soft fault does not.

The kernel tries to stay ahead of the system's demand for memory, so there is not necessarily a one-to-one correspondence between page-out events and page allocations by running processes. The goal of the system is to keep enough free

3. Ages ago, a second process known as "swapping" could occur by which all pages for a process were pushed out to disk at the same time. Today, demand paging is used in all cases.

memory handy that processes don't have to actually wait for a page-out each time they make a new allocation. If paging increases dramatically when the system is busy, it would probably benefit from more RAM.

Linux is still evolving rapidly, and its virtual memory system has not quite finished going through puberty—it's a little bit jumpy and a little bit awkward. You can tune the kernel's "swappiness" parameter (**/proc/sys/vm/swappiness**) to give the kernel a hint about how quickly it should make physical pages eligible to be reclaimed from a process in the event of a memory shortage. By default, this parameter has a value of 60. If you set it to 0, the kernel resorts to reclaiming pages that have been assigned to a process only when it has exhausted all other possibilities. If you set the parameter higher than 60 (the maximum value is 100), the kernel is more likely to reclaim pages. (If you find yourself tempted to modify this parameter, it's probably time to buy more RAM for the system.)

If the kernel fills up both RAM and swap, all VM has been exhausted. Linux uses an "out-of-memory killer" to handle this condition. This function selects and kills a process to free up memory. Although the kernel attempts to kill off the least important process on your system, running out of memory is always something to avoid. In this situation, it's likely that a substantial portion of the system's resources are being devoted to memory housekeeping rather than to useful work.

Analyzing memory usage

Two numbers summarize memory activity: the total amount of active virtual memory and the current paging rate. The first number tells you the total demand for memory, and the second suggests the proportion of that memory that is actively used. Your goal is to reduce activity or increase memory until paging remains at an acceptable level. Occasional paging is inevitable; don't try to eliminate it completely.

You can determine the amount of paging (swap) space that's currently in use. Run **swapon -s** on Linux, **swap -l** under Solaris and AIX, and **swapinfo** under HP-UX.

```
linux$ swapon -s
Filename    Type        Size       Used   Priority
/dev/hdb1   partition   4096532    0      -1
/dev/hda2   partition   4096564    0      -2
```

```
solaris$ swap -l
swapfile             dev  swapl   blocks   free
/dev/dsk/c0t0d0s1    32,1     16   164400  162960
```

```
hp-ux$ swapinfo
            Kb     Kb       Kb   PCT  START/        Kb
TYPE    AVAIL  USED     FREE  USED   LIMIT  RESERVE  PRI  NAME
dev   8388608     0  8388608    0%       0        -    1  /dev/vg00/lvol
```

swapinfo and **swapon** report usage in kilobytes, and **swap -l** uses 512-byte disk blocks. The sizes quoted by these programs do not include the contents of core memory, so you must compute the total amount of virtual memory yourself.

```
VM = size of real memory + amount of swap space used
```

On UNIX systems, paging statistics obtained with **vmstat** look similar to this output from Solaris:

```
solaris$ vmstat 5 5
procs      memory            page                    disk          faults
r b w   swap    free    re mf  pi po  fr  de  sr    s0 s6 s4 --   in   sy   cs
0 0 0 338216  10384   0  3   1  0   0   0   0     0  0  0  0  132  101  58
0 0 0 341784  11064   0 26   1  1   1   0   0     0  0  1  0  150  215  100
0 0 0 351752  12968   1 69   0  9   9   0   0     0  0  2  0  173  358  156
0 0 0 360240  14520   0 30   6  0   0   0   0     0  0  1  0  138  176  71
1 0 0 366648  15712   0 73   0  8   4   0   0     0  0 36  0  390  474  237
```

CPU information has been removed from this example. Under the procs heading is shown the number of processes that are immediately runnable, blocked on I/O, and runnable but swapped. If the value in the w column is ever nonzero, it is likely that the system's memory is pitifully inadequate relative to the current load.

The columns under the page heading give information about paging activity. All columns represent average values per second. Table 29.3 shows their meanings.

Table 29.3 Decoding guide for vmstat paging statistics

Column	Meaning
re	Number of pages reclaimed (rescued from the free list)
mf	Number of minor faults (minor meaning "small number of pages")
pi	Number of kilobytes paged in
po	Number of kilobytes paged out
fr	Number of kilobytes placed on the free list
de	Number of kilobytes of "predicted short-term memory shortfall"
sr	Number of pages scanned by the clock algorithm

The de column is the best indicator of serious memory problems. If it often jumps above 100, the machine is starved for memory. Unfortunately, some versions of **vmstat** don't show this number.

On Linux systems, paging statistics obtained with **vmstat** look like this:

```
linux$ vmstat 5 5
procs -----------memory---------- -swap- ---io--- --system-- -------cpu--------
 r  b   swpd   free   buff  cache   si  so   bi   bo   in   cs us sy id wa st
 5  0      0  66488  40328 597972    0   0  252   45 1042  278  3  4 93  1  0
 0  0      0  66364  40336 597972    0   0    0   37 1009  264  0  1 98  0  0
 0  0      0  66364  40344 597972    0   0    0    5 1011  252  1  1 98  0  0
 0  0      0  66364  40352 597972    0   0    0    3 1020  311  1  1 98  0  0
 0  0      0  66364  40360 597972    0   0    0   21 1067  507  1  3 96  0  0
```

As in the UNIX output, the number of processes that are immediately runnable and that are blocked on I/O are shown under the procs heading. Paging statistics are condensed to two columns, si and so, which represent pages swapped in and out, respectively.

Any apparent inconsistencies among the memory-related columns are for the most part illusory. Some columns count pages and others count kilobytes. All values are rounded averages. Furthermore, some are averages of scalar quantities and others are average deltas.

Use the si and so fields to evaluate the system's paging behavior. A page-in (si) does not necessarily represent a page being recovered from the swap area. It could be executable code being paged in from a filesystem or a copy-on-write page being duplicated, both of which are normal occurrences that do not necessarily indicate a shortage of memory. On the other hand, page-outs (so) always represent data written to disk after being forcibly ejected by the kernel.

If your system has a constant stream of page-outs, it's likely that you would benefit from more physical memory. But if paging happens only occasionally and does not produce annoying hiccups or user complaints, you can ignore it. If your system falls somewhere in the middle, further analysis should depend on whether you are trying to optimize for interactive performance (e.g., a workstation) or to configure a machine with many simultaneous users (e.g., a compute server).

On a traditional hard disk, you can figure that every 100 page-outs cause about one second of latency.[4] If 150 page-outs must occur to let you scroll a window, you will wait for about 1.5 seconds. A rule of thumb used by interface researchers is that an average user perceives the system to be "slow" when response times are longer than seven-tenths of a second.

Analyzing disk I/O

You can monitor disk performance with the **iostat** command. Like **vmstat**, it accepts optional arguments to specify an interval in seconds and a repetition count, and its first line of output is a summary since boot. Like **vmstat**, it also tells you how the CPU's time is being spent. Here is an example from Solaris:

```
solaris$ iostat 5 5
     tty          sd0           sd1           nfs1           cpu
tin  tout   kps  tps  serv   kps  tps  serv   kps  tps  serv   us  sy  wt  id
  0     1     5    1    18    14    2    20     0    0     0    0   0   0  99
  0    39     0    0     0     2    0    14     0    0     0    0   0   0 100
  2    26     3    0    13     8    1    21     0    0     0    0   0   0 100
  3   119     0    0     0    19    2    13     0    0     0    0   1   1  98
  1    16     5    1    19     0    0     0     0    0     0    0   0   0 100
```

4. We assume that about half of disk operations are page-outs.

Columns are divided into topics (in this case, five: tty, sd0, sd1, nfs1, and cpu), with the data for each topic presented in the fields beneath it. **iostat** output tends to be somewhat different on every system.

The tty topic presents data concerning terminals and pseudo-terminals. This information is basically uninteresting, although it might be useful for characterizing the throughput of a modem. The tin and tout columns give the average total number of characters input and output per second by all of the system's terminals.

Each hard disk has columns kps, tps, and serv, indicating kilobytes transferred per second, total transfers per second, and average "service times" (seek times, essentially) in milliseconds. One transfer request can include several sectors, so the ratio between kps and tps tells you whether there are a few large transfers or lots of small ones. Large transfers are more efficient. Calculation of seek times seems to work only on specific drives and sometimes gives bizarre values (the values in this example are reasonable).

iostat output on Linux, HP-UX, and AIX looks more like this:

```
aix$ iostat
...
Device:          tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
hdisk0          0.54         0.59         2.39     304483    1228123
hdisk1          0.34         0.27         0.42     140912     216218
hdisk2          0.01         0.02         0.05       5794      15320
hdisk3          0.00         0.00         0.00          0          0
```

Each hard disk has the columns tps, Blk_read/s, Blk_wrtn/s, Blk_read, and Blk_wrtn, indicating I/O transfers per second, blocks read per second, blocks written per second, total blocks read, and total blocks written.

Disk blocks are typically 1KiB in size, so you can readily determine the actual disk throughput in KiB/s. Transfers, on the other hand, are nebulously defined. One transfer request can include several logical I/O requests over several sectors, so this data is also mostly useful for identifying trends or irregular behavior.

The cost of seeking is the most important factor affecting disk drive performance. To a first approximation, the rotational speed of the disk and the speed of the bus to which the disk is connected to have relatively little impact. Modern disks can transfer hundreds of megabytes of data per second if they are read from contiguous sectors, but they can only perform about 100 to 300 seeks per second. If you transfer one sector per seek, you can easily realize less than 5% of the drive's peak throughput.

Seeks are more expensive when they make the heads travel a long distance. If you have a disk with several filesystem partitions and files are read from each partition in a random order, the heads must travel back and forth a long way to switch between partitions. On the other hand, files within a partition are relatively local to one another. When partitioning a new disk, consider the performance implications and put files that are accessed together in the same filesystem.

To really achieve maximum disk performance, you should put filesystems that are used together on different disks. Although the bus architecture and device drivers influence efficiency, most computers can manage multiple disks independently, thereby dramatically increasing throughput. For example, it is often worthwhile to split frequently accessed web server data and logs among multiple disks.

It's especially important to split the paging (swap) area among several disks if possible, since paging tends to slow down the entire system. Many systems can use both dedicated swap partitions and swap files on a formatted filesystem.

Some systems also let you set up multiple "memory-based filesystems," which are essentially the same thing as PC RAM disks. A special driver poses as a disk but actually stores data in high-speed memory. Many sites use a RAM disk for their **/tmp** filesystem or for other busy files such as web server logs or email spools. Using a RAM disk reduces the memory available for general use, but it makes the reading and writing of temporary files blindingly fast. It's generally a good deal.

*See page 144 for more information about **lsof** and **fuser**.*

The **lsof** command, which lists open files, and the **fuser** command, which shows the processes that are using a filesystem, can be helpful for isolating disk I/O performance issues. These commands show interactions between processes and filesystems, some of which may be unintended. For example, if an application is writing its log to the same device used for database logs, a disk bottleneck may result.

xdd: analyze disk subsystem performance

Modern storage systems can involve network or SAN-attached elements, RAID arrays, and other layers of abstraction. Consider the **xdd** tool for measuring and optimizing these complex systems. **xdd** is available under the GPL and runs on all of our example systems (not to mention Windows).

xdd measures subsystem I/O on single systems and on clusters of systems. It is well documented and yields accurate and reproducible performance measurements. You can read more about it at ioperformance.com.

sar: collect and report statistics over time

The **sar** command is a performance monitoring tool that has lingered through multiple UNIX and Linux epochs despite its somewhat obtuse command-line syntax. The original command has its roots in early AT&T UNIX.

At first glance, **sar** seems to display much the same information as **vmstat** and **iostat**. However, there's one important difference: **sar** can report on historical as well as current data.

*The Linux package that contains **sar** is called sysstat.*

Without options, the **sar** command reports CPU utilization for the day at 10-minute intervals since midnight, as shown below. This historical data collection is made possible by the **sal** script, which is part of the **sar** toolset and must be set up to run from **cron** at periodic intervals. **sar** stores the data it collects underneath the **/var/log** directory in a binary format.

```
linux$ sar
Linux 2.6.18-92.ELsmp (bajafur.atrust.com)  01/16/2010

12:00:01 AM       CPU     %user     %nice   %system   %iowait    %idle
12:10:01 AM       all      0.10      0.00      0.04      0.06     99.81
12:20:01 AM       all      0.04      0.00      0.03      0.05     99.88
12:30:01 AM       all      0.04      0.00      0.03      0.04     99.89
12:40:01 AM       all      0.09      0.00      0.03      0.05     99.83
12:50:01 AM       all      0.04      0.00      0.03      0.04     99.88
01:00:01 AM       all      0.05      0.00      0.03      0.04     99.88
```

In addition to CPU information, **sar** can also report on metrics such as disk and network activity. Use **sar -d** for a summary of this day's disk activity or **sar -n DEV** for network interface statistics. **sar -A** reports all available information.

sar has some limitations, but it's a good bet for quick-and-dirty historical information. If you're serious about making a long-term commitment to performance monitoring, we suggest that you set up a data collection and graphing platform such as Cacti. Cacti comes to us from the network management world, but it can actually graph arbitrary system metrics such as CPU and memory information. See page 886 for some additional comments on Cacti and an example of the graphs that it's capable of producing.

nmon and nmon_analyser: monitor in AIX

On AIX systems, **nmon** is the monitoring tool of choice. It is similar to **sar** in many ways.

Stephen Atkins of IBM developed a super-spreadsheet called **nmon_analyser** that processes the data collected by **nmon**. It's great for producing cleaned-up data as well as for creating presentation graphs. It analyzes data with more sophistication than does **sar**. For example, it can calculate weighted averages for hot-spot analysis and can integrate IBM and EMC disk performance information. Although **nmon_analyser** is not officially supported by IBM, you can find it at

ibm.com/developerworks/aix/library/au-nmon_analyser

Choosing a Linux I/O scheduler

Linux systems use an I/O scheduling algorithm to mediate between processes competing to perform disk I/O. The I/O scheduler massages the order and timing of disk requests to provide the best possible overall I/O performance for a given application or situation.

Four different scheduling algorithms are available in the Linux 2.6 kernel. You can take your pick. Unfortunately, the scheduling algorithm is set at boot time (with the **elevator=***algorithm* kernel argument), so it's not easy to change. The system's scheduling algorithm is usually specified in the GRUB boot loader's configuration file, **grub.conf**.

The available algorithms are

- Completely Fair Queuing (**elevator=cfq**): This is the default algorithm and is usually the best choice for general-purpose servers. It tries to evenly distribute access to I/O bandwidth. (If nothing else, the algorithm surely deserves an award for marketing: who could ever say no to a completely fair scheduler?)
- Deadline (**elevator=deadline**): This algorithm tries to minimize the latency for each request. It reorders requests to increase performance.
- NOOP (**elevator=noop**): This algorithm implements a simple FIFO queue. It assumes that I/O requests have already been optimized or reordered by the driver or will be optimized or reordered by the device (as might be done by an intelligent controller). This option may be the best choice in some SAN environments and is the best choice for SSD drives.

By determining which scheduling algorithm is most appropriate for your environment (you may need to run trials with each scheduler) you may be able to improve I/O performance.

oprofile: profile Linux systems in detail

oprofile is an incredibly powerful integrated system profiler for Linux systems running the 2.6 kernel or later. All components of a Linux system can be profiled: hardware and software interrupt handlers, kernel modules, the kernel itself, shared libraries, and applications.

If you have a lot of extra time on your hands and want to know exactly how your system resources are being used (down to the smallest level of detail), consider running **oprofile**. This tool is particularly useful if you are developing your own in-house applications or kernel code.

Both a kernel module and a set of user-level tools are included in the **oprofile** distribution, which is available for download at oprofile.sourceforge.net.

As of early 2010, a new system for tracing performance is on the horizon. Known as the performance events ("perf events") subsystem, it provides a level of instrumentation never before seen in the Linux kernel. This is likely to be the future of Linux performance profiling and is slated to eventually replace **oprofile**.

29.5 HELP! MY SYSTEM JUST GOT REALLY SLOW!

In previous sections, we've talked mostly about issues that relate to the average performance of a system. Solutions to these long-term concerns generally take the form of configuration adjustments or upgrades.

However, you will find that even properly configured systems are sometimes more sluggish than usual. Luckily, transient problems are often easy to diagnose. Most of the time, they are caused by a greedy process that is simply consuming so

much CPU power, disk, or network bandwidth that other processes are affected. On occasion, malicious processes hog available resources to intentionally slow a system or network, a scheme known as a "denial of service" or DOS attack.

You can often tell which resource is being hogged without even running a diagnostic command. If the system feels "sticky" or you hear the disk going crazy, the problem is most likely a disk bandwidth or memory shortfall.[5] If the system feels "sluggish" (everything takes a long time, and applications can't be "warmed up"), the problem may lie with the CPU load.

The first step in diagnosis is to run **ps auxww** (**ps -elf** on Solaris and HP-UX) or **top** to look for obvious runaway processes. Any process that's using more than 50% of the CPU is likely to be at fault. If no single process is getting an inordinate share of the CPU, check to see how many processes are getting at least 10%. If you snag more than two or three (don't count **ps** itself), the load average is likely to be quite high. This is, in itself, a cause of poor performance. Check the load average with **uptime**, and use **vmstat** or **top** to check whether the CPU is ever idle.

If no CPU contention is evident, run **vmstat** to see how much paging is going on. All disk activity is interesting: a lot of page-outs may indicate contention for memory, and disk traffic in the absence of paging may mean that a process is monopolizing the disk by constantly reading or writing files.

There's no direct way to tie disk operations to processes, but **ps** can narrow down the possible suspects for you. Any process that is generating disk traffic must be using some amount of CPU time. You can usually make an educated guess about which of the active processes is the true culprit.[6] Use **kill -STOP** to suspend the process and test your theory.

Suppose you do find that a particular process is at fault—what should you do? Usually, nothing. Some operations just require a lot of resources and are bound to slow down the system. It doesn't necessarily mean that they're illegitimate. It is sometimes useful to **renice** an obtrusive process that is CPU-bound, however.

Sometimes, application tuning can dramatically reduce a program's demand for CPU resources; this effect is especially visible with custom network server software such as web applications.

Processes that are disk or memory hogs can't be dealt with so easily. **renice** generally does not help. You do have the option of killing or stopping the process, but we recommend against this if the situation does not constitute an emergency. As with CPU pigs, you can use the low-tech solution of asking the owner to run the process later.

5. That is, it takes a long time to switch between applications, but performance is acceptable when an application is repeating a simple task.
6. A large virtual address space or resident set used to be a suspicious sign, but shared libraries have made these numbers less useful. **ps** is not very smart about separating system-wide shared library overhead from the address spaces of individual processes. Many processes wrongly appear to have tens of megabytes of active memory.

The kernel allows a process to restrict its own use of physical memory by calling the **setrlimit** system call.[7] This facility is also available in the C shell through the built-in **limit** command. For example, the command

```
% limit memoryuse 32m
```

causes all subsequent commands that the user runs to have their use of physical memory limited to 32MiB (Solaris uses **memorysize** rather than **memoryuse**). This feature is roughly equivalent to **renice** for memory-bound processes.

If a runaway process doesn't seem to be the source of poor performance, investigate two other possible causes. The first is an overloaded network. Many programs are so intimately bound up with the network that it's hard to tell where system performance ends and network performance begins. See Chapter 21 for more information about the tools used to monitor networks.

Some network overloading problems are hard to diagnose because they come and go very quickly. For example, if every machine on the network runs a network-related program out of **cron** at a particular time each day, there will often be a brief but dramatic glitch. Every machine on the net will hang for five seconds, and then the problem will disappear as quickly as it came.

Server-related delays are another possible cause of performance crises. UNIX and Linux systems are constantly consulting remote servers for NFS, Kerberos, DNS, and any of a dozen other facilities. If a server is dead or some other problem makes the server expensive to communicate with, the effects ripple back through client systems.

For example, on a busy system, some process may use the **gethostent** library routine every few seconds or so. If a DNS glitch makes this routine take two seconds to complete, you will likely perceive a difference in overall performance. DNS forward and reverse lookup configuration problems are responsible for a surprising number of server performance issues.

29.6 RECOMMENDED READING

COCKCROFT, ADRIAN, AND BILL WALKER. *Capacity Planning for Internet Services.* Upper Saddle River, NJ: Prentice Hall. 2001.

DREPPER, ULRICH. *What Every Programmer Should Know about Memory.* lwn.net/Articles/250967.

EZOLT, PHILLIP G. *Optimizing Linux Performance.* Upper Saddle River, NJ: Prentice Hall PTR, 2005.

JOHNSON, S., ET AL. *Performance Tuning for Linux Servers.* Indianapolis, IN: IBM Press, 2005.

7. More granular resource management can be achieved through the Class-based Kernel Resource Management functionality; see ckrm.sourceforge.net.