PEARSON SERIES IN COMPUTER NETWORKING AND DISTRIBUTED SYSTEMS

THIRD EDITION

NETWORK SECURITY

PRIVATE Communication in a PUBLIC World

NEW CONTENT
Quantum computing, post-quantum algorithms, multiparty computation, fully homomorphic encryption, and more!

CHARLIE KAUFMAN • RADIA PERLMAN
MIKE SPECINER • RAY PERLNER

Network Security

PRIVATE Communication in a PUBLIC World

THIRD EDITION

Charlie Kaufman • Radia Perlman
Mike Speciner • Ray Perlner

Addison-Wesley

Boston • Columbus • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Library of Congress Control Number: 2022941173

ISBN-13: 978-0-13-664360-9
ISBN-10: 0-13-664360-4

ScoutAutomatedPrintCode

Si spy net work, big fedjaw iog link kyxogy

CONTENTS

ACKNOWLEDGMENTS

Despite the controversies that crop up around security issues, it has been our experience that people in the security community are generally generous with their wisdom and time. It's always a little scary thanking specific people, for fear we'll leave someone out, but leaving everyone out seems wrong. It's not even the fair thing to do, since some people would be more egregiously wronged by being left out than others.

We'd like to thank the reviewers of various chapters of this book, which include Luís T. A. N. Brandão, Bill Burr, Lisa Carnahan, Lily Chen, David Cooper, Joan Daemen, Michael Davidson, Morrie Dworkin, Ned Freed, Simson Garfinkel, Jonathan Katz, John Kelsey, Yi-Kai Liu, Kerry McKay, Carl Miller, Dustin Moody, Nicky Mouha, Rene Peralta, Andy Regenscheid, Scott Rose, Seth Rothschild, Matt Scholl, Katherine Sharpless, Jim St. Pierrez, Kevin Stine, Sheryl Taylor, Meltem Sönmez Turan, and Tatu Ylönen.

Other people who have been helpful with answering questions, especially on previous editions of this book, include Eric Rescorla, Al Eldridge, Hilarie Orman, Tom Wu, Kevin Fu, Amir Herzberg, Mark Davis, Marshall Rose, Joe Tardo, Joe Pato, Seth Proctor, Timothy Spiller, Tom Rice, Kristen McIntyre, Gary Winiger, Dan Harkins, Peter Memishian, Jeff Schiller, Burt Kaliski, Tony Lauck, Phil Karn, Ron Rivest, Steve Crocker, Steve Kent, John Linn, Steve Hanna, Jim Bidzos, Dave Jablon, Ted Ts'o, Matthew Barnes, Keith McCloughrie, Jeffrey Case, Walt Tuvell, Cliff Van Dyke, Kathrin Winkler, Philippe Auphelle, Sig Handelman, Phillip Hallam-Baker, Uri Blumenthal, Serge Vaudenay, and Boyd Roberts.

And of course we thank you, our reader. We welcome your comments and suggestions. Compliments are always welcome. We hope to update the book periodically, so if there are topics you wish we'd covered or errors you'd like us to correct, let us know. Errata can be found at www.informit.com/title/9780136643609.

Our current email addresses are charliekaufman@outlook.com, radia@alum.mit.edu, ms@alum.mit.edu, and rperlner@gmail.com.

ABOUT THE AUTHORS

Charlie Kaufman is currently Security Architect for Dell Storage Systems. Previously, he was the Security Architect for Microsoft Azure and before that for Lotus Notes. He has contributed to a number of IETF standards efforts including IPsec, S/MIME, and DNSSEC and served as a member of the Internet Architecture Board. He served on the National Academy of Sciences expert panel that wrote the book *Trust In Cyberspace*.

Radia Perlman is currently a Fellow at Dell Technologies. She is known for her contributions to bridging (spanning tree algorithm), routing (link state routing), and security (distributed systems robust despite malicious participants). She's the author of *Interconnections: Bridges, Routers, Switches, and Internetworking Protocol*. She's been elected to the National Academy of Engineering, the National Inventors Hall of Fame, the Internet Hall of Fame, and awarded lifetime achievement awards from Usenix and ACM's SIGCOMM. She has a PhD in computer science from MIT.

Mike Speciner is an MIT-trained technologist with expertise in mathematics, physics, and computer science. He currently serves as CTO and cofounder of The Singing Torah. His hobby is writing software for educational purposes in various common and obscure programming languages.

Ray Perlner is a Mathematician in the Cryptographic Technology Group of the National Institute of Standards and Technology. He has over a dozen research papers focusing primarily on post-quantum cryptography. He has degrees in both physics and mathematics from MIT.

1 *Introduction*

It was a dark and stormy night. Somewhere in the distance a dog howled. A shiny object caught Alice's eye. A diamond cufflink! Only one person in the household could afford diamond cufflinks! So it was the butler, after all! Alice had to warn Bob. But how could she get a message to him without alerting the butler? If she phoned Bob, the butler might listen on an extension. If she sent a carrier pigeon out the window with the message taped to its foot, how would Bob know it was Alice that was sending the message and not Trudy attempting to frame the butler because he spurned her advances?

That's what this book is about. Not much character development for Alice and Bob, we're afraid; nor do we really get to know the butler. But we do discuss how to communicate securely over an insecure medium.

What do we mean by "communicating securely"? Alice should be able to send a message to Bob that only Bob can understand, even though Alice can't avoid having others see what she sends. When Bob receives a message, he should be able to know for certain that it was Alice who sent the message, and that nobody tampered with the contents of the message in the time between when Alice launched the message and Bob received it.

What do we mean by an "insecure medium"? Well, in some dictionary or another, under the definition of "insecure medium" should be a picture of the Internet. The world is evolving towards interconnecting every computer, household appliance, automobile, child's toy, and embedded medical device, all into some wonderful global internetwork. How wonderful! You'd be able to control your nuclear power plant with simple commands sent across the network while you were vacationing in Fiji. Or sunny Havana. Or historic Pyongyang. Inside the network the world is scary. There are links that eavesdroppers can listen in on. Information needs to be forwarded through packet switches, and these switches can be reprogrammed to listen to or modify data in transit.

The situation might seem hopeless, but we may yet be saved by the magic of cryptography, which can take a message and transform it into a bunch of numbers known as ciphertext. The ciphertext is unintelligible gibberish except to someone who knows the secret to reversing the transformation. Cryptography allows us to disguise our data so that eavesdroppers gain no information from listening to the information as transmitted. Cryptography also allows us to create an unforgeable message and detect if it has been modified in transit. One method of accomplishing this is with a **digital signature**, a number associated with a message and its sender that can be verified as authentic by others, but can only be generated by the sender. This should seem astonishing. How can there be a number you can verify but not generate? A person's handwritten signature can (more or less) only be generated by that person, though it can (more or less) be verified by others. But it would seem as if a number shouldn't be hard to generate, especially if it can be verified. Theoretically, you could generate someone's digital signature by trying lots of numbers and testing each one until one passed the verification test. But with the size of the numbers used, it would take too much compute time (for instance, several universe lifetimes) to generate the signature that way. So a digital signature has the same property as a handwritten signature (theoretically) has, in that it can only be generated by one person but can be verified by lots of people. But a digital signature does more than a handwritten signature. Since the digital signature depends on the contents of the message, if someone alters the message the signature will no longer be correct, and the tampering will be detected. This will all become clear if you read Chapter 2 *Introduction to Cryptography*.

Cryptography is a major theme in this book, not because cryptography is intrinsically interesting (which it is), but because many of the security features people want in a computer network can best be provided through cryptography.

1.1 OPINIONS, PRODUCTS

Opinions expressed are those of the authors alone (and possibly not even agreed upon by all the authors). Opinions do not necessarily represent the views of any of the authors' past, current, or future employers. Any mention of commercial products or reference to commercial organizations is for information only. It does not imply recommendation or endorsement by NIST or any of the current, future, or prior organizations employing any of the authors.

1.2 ROADMAP TO THE BOOK

We aim to make this book comprehensible to engineers, giving intuition about designs. But readability doesn't mean lack of technical depth. We try to go beyond the information one might find in specifications to give insight into the designs. Given that specifications are easily available on the web today, we do not give exact packet formats.

This book should be usable as a textbook at either the undergraduate or graduate level. Most of the chapters have homework problems at the end. And to make life easier for professors who want to use the book, we will provide slides and an answer manual (to the professors using the book) for many of the chapters. Even if you are not taking a class, you might want to do the homework problems. This book should be understandable to anyone with technical curiosity, a sense of humor*, and a good night's sleep in the recent past. The chapters are:

- **Chapter 1** ***Introduction***: This gives an overview of the chapters and some network basics.
- **Chapter 2** ***Introduction to Cryptography***: This explains the cryptographic principles which will be covered in more detail in later chapters.
- **Chapter 3** ***Secret Key Cryptography***: This chapter explains the uses of secret key cryptographic algorithms and describes how cryptographers create these algorithm.
- **Chapter 4** ***Modes of Operation***: Since most secret key algorithms encrypt a fairly small (*e.g.*, 128 bits) block, this chapter explains various algorithms to efficiently and securely encrypt arbitrarily large amounts of data.
- **Chapter 5** ***Cryptographic Hashes***: This chapter explains what hashes are used for and intuition into the methods by which cryptographers create secure and efficient hashes.
- **Chapter 6** ***First-Generation Public Key Algorithms***: This chapter describes the designs of the current widely deployed public key algorithms. Unfortunately, these would be insecure if the world were able to create a sufficiently large quantum computer. So the world will migrate to different public key algorithms that we'll describe in Chapter 8 *Post-Quantum Cryptography*.
- **Chapter 7** ***Quantum Computing***: This chapter gives an intuitive understanding of how a quantum computer differs from a classical computer, as well as explaining the intuition behind the two major cryptography-relevant quantum algorithms (Grover's and Shor's).
- **Chapter 8** ***Post-Quantum Cryptography***: This describes the types of math problems that would remain difficult to solve even if there were quantum computers, how to turn them into public key algorithms, and various optimizations that can make them efficient.
- **Chapter 9** ***Authentication of People***: This chapter describes the challenges involved in authenticating humans, and various types of technology that are or could be deployed.
- **Chapter 10** ***Trusted Intermediaries***: This chapter describes technologies for distributing cryptographic keys. It also talks about trust model issues in today's deployed designs.
- **Chapter 11** ***Communication Session Establishment***: This chapter describes conceptual issues in doing mutual authentication handshakes and establishing secure sessions.
- **Chapter 12** ***IPsec***: This chapter goes into detail about the design of IPsec.
- **Chapter 13** ***SSL/TLS and SSH***: This chapter goes into detail about the design of SSL/TLS and SSH.

*Although a sense of humor is not strictly necessary for understanding the book, it is an important characteristic to have in general.

- **Chapter 14** ***Electronic Mail Security***: This chapter describes various issues and solutions involved in electronic mail.
- **Chapter 15** ***Electronic Money***: This chapter describes various goals of electronic money and various technologies to address them. It covers cryptocurrencies and anonymous cash.
- **Chapter 16** ***Cryptographic Tricks***: This describes various exotic technologies such as secure multiparty computation and homomorphic encryption, as well as widely used technologies such as secret sharing.
- **Chapter 17** ***Folklore***: This chapter gives a summary of some of the design lessons discussed in the rest of the book and also describes some common misconceptions.
- **Glossary**: We define many of the terms we use in the book.
- **Math**: This provides more in-depth coverage of the mathematics used in the rest of the book. The appendix is written solely by Mike Speciner. It's not essential for appreciating the rest of the book. It's a sample of the content in Mike's github repositories https://github.com/ms0/.

1.3 TERMINOLOGY

Computer science is filled with ill-defined terminology used by different authors in conflicting ways. Some people take terminology very seriously, and once they start to use a certain word in a certain way, are extremely offended if the rest of the world does not follow.

> *When I use a word, it means just what I choose it to mean—neither more nor less.*
>
> —Humpty Dumpty (in *Through the Looking Glass*)

Some terminology we feel fairly strongly about. We do *not* use the term *hacker* to describe the vandals that break into computer systems. These criminals call themselves hackers, and that is how they got the name. But they do not deserve the name. True hackers are master programmers, incorruptibly honest, unmotivated by money, and careful not to harm anyone. The criminals termed "hackers" are not brilliant and accomplished. It is really too bad that they not only steal money, people's time, and worse, but they've also stolen a beautiful word that had been used to describe some remarkable and wonderful people. We instead use words like *intruder*, *bad guy*, and *impostor*.

We grappled with the terms *secret key* and *public key* cryptography. Often in the security literature the terms *symmetric* and *asymmetric* are used instead of *secret* and *public*. When we say *secret key*, we mean a key that is used both for encryption and decryption. When we say *public key*, we are referring to a key pair consisting of a public key (used for encryption or signature verification) and a private key (used for decryption or signing). Using the terms *public key* and *private key* is occasionally regrettable because both the words *public* and *private* start with "p".

We use the term *privacy* when referring to the desire to keep communication from being seen by anyone other than the intended recipients. Some people in the security community avoid the term *privacy* because they feel its meaning has been corrupted to mean *the right to know*, because in some countries there are laws known as *privacy laws* that state that citizens have the right to see records kept about them. *Privacy* also tends to be used when referring to keeping personal information about people from being collected and misused. The security community also avoids the use of the word *secrecy*, because *secret* has special meaning within the military context, and they feel it would be confusing to talk about the secrecy of a message that was not actually labeled top secret or secret. The term most commonly used in the security community for keeping communication from being seen is *confidentiality*. We find that strange because *confidential*, like *secret*, is a security label, and the security community should have scorned use of *confidential*, too. In the first edition, we chose not to use *confidentiality* because we felt it had too many syllables, and saw no reason not to use *privacy*. For the second edition we reconsidered this decision, and were about to change all use of *privacy* to *confidentiality* until one of us pointed out we'd have to change the book title to something like *Network Security: Confidential Communication in a Non-Confidential World*, at which point we decided to stick with *privacy*.

Speaker: *Isn't it terrifying that on the Internet we have no privacy?*
Heckler$_1$: *You mean* confidentiality. *Get your terms straight.*
Heckler$_2$: *Why do security types insist on inventing their own language?*
Heckler$_3$: *It's a denial-of-service attack.*
—Overheard at gathering of security types

We often refer to things involved in a conversation by name; for instance, *Alice* and *Bob*, whether the things are people or computers. This is a convenient way of making descriptions unambiguous with relatively few words, since the pronoun *she* can be used for Alice, and *he* can be used for Bob. It also avoids lengthy inter-author arguments about whether to use the politically incorrect *he*, a confusing *she*, an awkward *he/she* or *(s)he*, an ungrammatical *they*, an impersonal *it*, or an awkward rewriting to avoid the problem. We remain slightly worried that people will assume when we've named things with human names that we are always referring to people. Assume Alice, Bob, and the rest of the gang may be computers unless we specifically say something like *the user Alice*, in which case we're talking about a human.

When we need a name for a bad guy, we usually choose *Trudy* (since it sounds like *intruder*) or *Eve* (since it sounds like *eavesdropper*) or *Mallory* (since it sounds like *malice*). Everyone would assume Alice, Eve, and Trudy are she, and Bob is he. For inclusivity, we wanted at least one of the evil characters to be male, and we chose Mallory as the name of a male evildoer. Mallory can be used for either gender, and is gaining more popularity as a female name, but when we use Mallory we will assume Mallory is male and use the pronoun *he*.

With a name like yours, you might be any shape, almost.
—Humpty Dumpty to Alice (in *Through the Looking Glass)*

Occasionally, one of the four of us authors will want to make a personal comment. In that case we use *I* or *me* with a subscript. When it's a comment that we all agree with, or that we managed to slip past me_3 (the rest of us are wimpier), we use the term *we*.

1.4 NOTATION

We use the symbol $\oplus$ (pronounced *ex-or*) for bitwise-exclusive-or. We use the symbol | for concatenation. We denote encryption with curly brackets followed by the key with which something was encrypted, as in $\{message\}K$, which means *message* is encrypted with *K*. We denote a signature with square brackets followed by the key, as in $[message]_{\text{Bob}}$. Sometimes the key is a subscript, and sometimes not. There is no deep meaning to that. Honestly, it's that sometimes we are using a key that has subscripts, such as K_{Alice}, and the formatting tool we are using makes it very difficult to have a subscripted subscript.

1.5 CRYPTOGRAPHICALLY PROTECTED SESSIONS

When Alice and Bob use modern cryptography and protocols, such as IPsec (Chapter 12) or TLS (Chapter 13), they first exchange a few messages in which they establish session secrets. These session secrets allow them to encrypt and integrity-protect their conversation. Although their physical connectivity is a path across the Internet, once Alice and Bob create the protected session, data that they send to each other is as trustworthy as if they had a private physically protected link.

There are various terms for this type of protected session. We will usually refer to it as a **secure session**. It is considered good security practice to use several cryptographic keys in a secure session between Alice and Bob. For example, there might be different session keys for

- encryption of Alice to Bob traffic,
- encryption of Bob to Alice traffic,

- integrity protection of Alice to Bob traffic, and
- integrity protection of Bob to Alice traffic.

Alice and Bob will each have a database describing their current secure sessions. The information in the database will include information such as how the session will be identified on incoming data, who is on the other end of the session, which cryptographic algorithms are to be used, and the sequence numbers for data to be sent or received on the session.

1.6 ACTIVE AND PASSIVE ATTACKS

A **passive attack** is one in which the intruder eavesdrops but does not modify the message stream in any way. An **active attack** is one in which the intruder may transmit messages, replay old messages, modify messages in transit, or delete or delay selected messages in transit. Passive attacks are less risky for the attacker, because it is tricky to detect or prove someone has eavesdropped. If the attacker is not on the path between Alice and Bob, the passive attack can be done by having an accomplice router make copies of the traffic and send them to the attacker, who can then analyze the data later, in private.

A typical active attack is one in which an intruder impersonates one end of the conversation, or acts as a **meddler-in-the-middle** (**MITM**). (Note that the acronym MITM used to be expanded to be man-in-the-middle, but the industry is trying to move to more inclusive language. In this case, it is acknowledging that being annoying is not gender-specific.) A MITM attack is where an active attacker, say, Trudy, acts as a relay between two parties (Alice and Bob), and rather than simply forwarding messages between Alice and Bob, modifies, deletes, or inserts messages. If Trudy were faithfully forwarding messages, she could be acting as a passive eavesdropper, or she could be a correctly functioning router.

If Alice communicates with Bob using a secure session protocol with strong cryptographic protection, Trudy would gain no information by eavesdropping and would not be able to modify messages without being detected.

However, Trudy might be able to impersonate Bob's IP address to Alice, tricking Alice into establishing a secure session between Alice and Trudy. Then Trudy can simultaneously impersonate Alice to Bob, and establish a secure session between Trudy and Bob. (See Figure 1-1.)

$$\begin{tikzcd} \text{Alice} \arrow[r, leftrightarrow, "\text{shared key } K_{\text{A-T}}"'] & \text{Trudy} \arrow[r, leftrightarrow, "\text{shared key } K_{\text{T-B}}"'] & \text{Bob} \end{tikzcd}$$

Figure 1-1. MITM Attack

Alice and Bob will think they are talking to each other, but in fact they are each talking to Trudy. Data sent by Alice to Bob will be decrypted by Trudy using the session secret for the Alice-Trudy secure session, and encrypted for Bob with the session secret for the Trudy-Bob secure session. It is difficult for Alice and Bob to know that they have a MITM. Alice could attempt to make sure she's really talking to Bob by asking questions such as "What did I order when we first met for dinner?", but Trudy can forward the questions and answers. We will explain in §11.6 *Detecting MITM* how Alice and Bob can detect a MITM. And as we will explain in later chapters, if Alice has credentials for Bob that Trudy cannot impersonate, Alice and Bob can prevent a MITM.

1.7 LEGAL ISSUES

The legal aspects of cryptography are fascinating, but the picture changes quickly, and we are certainly not experts in law. Although it pains us to say it, if you're going to build anything involving cryptography, talk to a lawyer. The combination of patents and export controls slowed down deployment of cryptographically secure networking, and caused strange technical choices.

1.7.1 Patents

One legal issue that affects the choice of security mechanisms is patents. Most cryptographic techniques were covered by patents and historically this has slowed their deployment. One of the important criteria for NIST's selection of algorithms (such as AES [§3.7 *Advanced Encryption Standard (AES)*], SHA-3 [§5.6.2 *Construction of SHA-3*], and post-quantum algorithms [Chapter 8 *Post-Quantum Cryptography*]) is whether they are royalty-free.

The widely deployed RSA algorithm (see §6.3) was developed at MIT, and under the terms of MIT's funding at the time, there were no license fees for U.S. government use. It was only patented in the U.S., and licensing was controlled by one company, which claimed that the Hellman-Merkle patent also covered RSA, and that patent is international. Interpretation of patent rights varies by country, so the legal issues were complex. At any rate, the last patent on RSA ran out on 20 September 2000. There were many parties on that day.

"I don't know what you mean by your way,*" said the Queen: "all the ways about here belong to me…"*

—Through the Looking Glass

To avoid large licensing fees, many protocol standards used DSA (see §6.5) instead of RSA. Although in most respects DSA is technically inferior to RSA, when first announced it was advertised that DSA would be freely licensable so it would not be necessary to reach agreement with the RSA-licensing company. But the company claimed Hellman-Merkle covered all public key cryptography, and strengthened its position by acquiring rights to a patent by Schnorr that was closely related to DSA. Until the patents expired (and luckily the relevant patents have expired), the situation was murky.

1.7.2 Government Regulations

Mary had a little key
(It's all she could export)
And all the email that she sent
Was opened at the Fort.
—Ron Rivest

The U.S. government (as well as other governments) used to impose severe restrictions on export of encryption. This caused much bitterness in the computer industry and led to some fascinating technical designs so that domestic products, which were legally allowed to use strong encryption, could use strong encryption where possible, and yet interoperate with exportable products that were not allowed to use strong encryption. Although U.S. companies still need permission from the Department of Commerce to export products containing cryptography, since around the year 2000 it has been easy to get products approved.

Additionally, even today, some countries have usage controls, so even if it were legal to export a product, it might not be legally usable inside some other country. For instance, some countries have developed their own cryptographic algorithms, and they want all their citizens to use those. Most of the reason for these sorts of rules is so that a government can't be prevented from accessing data, for instance, for law enforcement purposes.

1.8 SOME NETWORK BASICS

Although I_2 get a bit frustrated with teaching network concepts as if TCP/IP is the only way, or the best way to design a network, this is what today's Internet is built with, so we need to understand some of the details. Here is a brief introduction.

1.8.1 Network Layers

A good way of thinking about networking concepts is with *layers*. The concept of a layer is that inside a node, there are interfaces to adjacent layers (the layer above or the layer below). Between nodes, there are protocols for talking to peer layers. The actual protocol inside a layer can in theory be replaced by a layer that gives similar functionality to the adjacent layers. Although layers are a good way to learn about networks, deployed networks do not cleanly follow a layering model. Layers often use data associated with layers other than peer layers or adjacent layers. Layers are often subdivided into more layers, and an implementation might merge layers. ISO (International Organization for Standardization) defined a model with seven layers. The bottom layers look like this:

- Layer 1, **physical layer**. Defines how to send a stream of bits to a neighbor node (neighbors reside on the same link).
- Layer 2, **data link layer**. Defines how to structure a string of bits (provided by layer 1) into packets between neighbor nodes. This requires using the stream of bits to signal information such as "this is the beginning of a packet", "this is the end of a packet", and an integrity check.
- Layer 3, **network layer**. This allows a source node to send a packet of information across many links. The source adds header information to a packet to let the network know where to deliver the packet. This is analogous to putting a postal message inside an envelope, and writing the destination on the envelope. A network will consist of many links. Nodes known as *routers* or *switches* forward between links. Such nodes are connected to two or more links. They have a table known as a *forwarding table* that tells them which link to forward on, to get closer to the destination. Usually, network addresses are assigned hierarchically, so that a bunch of addresses can be summarized in one forwarding entry. This is analogous to the post office only needing to look at the destination country, and then once inside that country, forwarding towards the state, and once inside the state, forwarding to the destination city, etc. The usual protocol deployed in the Internet today for layer 3 is IP (Internet Protocol), which basically consists of adding a header to a packet identifying the source and destination, a hop count (so the network can discard packets that are looping), and other information. There are two versions of IP. IPv4 has 32-bit addresses. IPv6 has 128-bit addresses. One extra piece of information in the IP header is the 16-bit "protocol type", which indicates which layer 4 protocol is sending the data.
- Layer 4, **transport layer**. This is information that is put in by the source, and interpreted at the destination. The service provided by TCP (Transmission Control Protocol, RFC 793) to the layer above it consists of accepting a stream of bytes at the source, and delivering the stream of bytes to the layer above TCP at the destination, without loss or duplication. To accomplish this, TCP at the sender numbers bytes; TCP at the destination uses the sequence numbers to acknowledge receipt of data, reorder data that has arrived out of sequence, and

ask for retransmission of lost data. UDP (User Datagram Protocol, RFC 768) is another layer 4 protocol that does not worry about lost or reordered data. Many processes in the layer above TCP or UDP will be reachable at the same IP address, so both UDP and TCP headers include **ports** (one for source, and one for destination), which tell the destination which process should receive the data.

1.8.2 TCP and UDP Ports

There are two 16-bit fields in TCP and UDP—a source port and a destination port**.** Typically an application on a server will be reachable at a "well-known port", meaning that the port is specified in the protocol. If a client wants to reach that application at a server, the protocol type field in the IP header will be either TCP (6) or UDP (17) and the destination port field in the layer 4 header (TCP or UDP in this case) will be the well-known port for that application. For example, HTTP is at port 80, and HTTPS is at port 443. The source port will usually be a dynamically assigned port (49152 through 65535).

1.8.3 DNS (Domain Name System)

Another aspect of Internet networking we will be discussing is DNS. It is basically a distributed directory that maps DNS names (*e.g.*, example.com) to IP addresses. DNS names are hierarchical. A simple way to think of DNS is that for each level in the DNS name (*e.g.*, root, .org, .com, example.com) there is a server that keeps a directory associated with names in that level. The root would have a directory that allows looking up servers for each of the top-level domains (TLDs) (*e.g.*, .org, .com, .gov, .tv). There are currently over a thousand TLDs, so the root would have information associated with each of those TLDs in its database. In general, to find a DNS name, a node starts at the root, finds the server that holds the directory for the next level down, and keeps going until it gets to the server that stores information about the actual name. There are several advantages to DNS being hierarchical.

- Someone that wishes to purchase a DNS name has a choice of organizations from which to purchase a name. If a name is purchased from the organization managing names in the TLD .org, the purchased name will be of the form example.org. If you purchase the name example.org, you can then name anything that would be below that name in the DNS hierarchy, such as xyz.example.com or labs.xyz.example.com.
- The DNS database will not become unmanageably large, because no organization needs to keep the entire DNS database. In fact, nobody knows how many names are in the DNS database.

- It is fine to have the same lower level name in multiple databases. For instance, there is no problem with there being DNS names example.com and example.org.

1.8.4 HTTP and URLs

When we access things on the web, we use a protocol known as HTTP (hypertext transfer protocol). HTTP allows specifying more than a DNS name; it allows specifying a particular web page at the service with a DNS name. The URL (uniform resource locator) is the address of the web page. The URL contains a DNS name of the service, followed by additional information that is interpreted solely by the server that receives the request. The additional information might be, for instance, the directory path at the destination server that finds the information to construct the page being requested.

Sometimes humans type URLs, but usually URLs are displayed as links in a webpage that can be clicked on. It is common for people to do an Internet search (*e.g.*, using Google or Bing) for something, and then click on choices. URLs can be very long and ugly, and people usually don't look at the URL they click on. Often the web page that displays a link does not display the actual URL. Mousing over the link will sometimes show the human a URL. Unfortunately, the web page can choose what to display on the page as the link, and what to display the mouse-over link as. These can be different from the actual URL that will be followed if the link is clicked on. For example, on a web page, a clickable link (which is usually displayed in a different color), might display as "click here for information", and if a suspicious user moused-over the link, it might display "http://www.example.com/information", but if the user clicks on the link, the malicious webpage could send them to any URL, *e.g.*, http://www.rentahitman.com.

The two main HTTP request types are GET and POST. GET is for reading a web page and POST is for sending information to a web server. The response contains information such as the content requested and status information (such as "OK" or "not found" or "unauthorized"). One status that might be included in a response is a redirect. This informs the browser that it should go to a different URL. The browser will then go to the new URL, as if the user had clicked on a link.

1.8.5 Web Cookies

If a client is browsing content that requires authentication and access control, or is accumulating information such as items in a virtual shopping basket to be purchased when the user is finished browsing the on-line catalog, the information for that session needs to be kept somewhere. But HTTP is stateless. Each request/response interaction is allowed to take place over a fresh TCP connection. The cookie mechanism enables the server to maintain context across many request/response interactions. A **cookie** is a piece of data sent to the client by the server in response to an HTTP request. The cookie need not be interpreted by the client. Instead, the client keeps a list of DNS names and cookies it has received from a server with that DNS name. If the client made a request at example.com, and example.com sent a cookie, the client will remember (example.com: cookie) in its cookie list. When the client next makes an HTTP request to example.com, it searches its cookie database for any cookies received from example.com, and includes those cookies in its HTTP request.

The cookie might contain all the relevant information about a user, or the server might keep a database with this information. In that case, the cookie only needs to contain the user's identity (so the server can locate that user in its database), along with proof that the user has already authenticated to the server. For example, if Alice has authenticated to Bob, Bob could send Alice a cookie consisting of some function of the name "Alice" and a secret that only Bob knows. A cookie will be cryptographically protected by the server in various ways. It might be encrypted with a key that only the server knows. It might contain information that only allows the cookie to be used from a specific machine. And it is almost always protected when transmitted across the network because the client and server will be communicating over a secure session.

1.9 NAMES FOR HUMANS

It is sometimes important that an identifier for a human be unique, but it is not important for a human to have a *single* unique name. Humans have many unique identifiers. For example, an email address, a telephone number, or a username specific to a website. In theory, nobody but the human needs to know that the various identities refer to the same human, but, unfortunately, it has become easy for organizations to correlate various identities. Also, it is not uncommon for family members or close friends to share an account, so an email address or username at a website might actually be multiple humans, but we will ignore that issue.

Human names are problematic. Consider email addresses. Typically, companies let the first John Smith use the name John@companyname for his email address, and then perhaps the next one will be Smith@companyname, and the next one JSmith@companyname, and the next one has to start using middle initials. Then, to send to your colleague John Smith, you have to do the best you can to figure out which email address in the company directory is the one you want, based on various attributes (such as their location or their job title, if you are lucky enough to have this information included in the directory). There will be lots of confusion when one John Smith gets messages intended for a different John Smith. This is a problem for both the John Smith that is mistakenly sent the email, as well as the John Smith who was the intended recipient of the email. Usually, a person can quickly delete spam, but with a name like John Smith, irrelevant-looking email might actually be important email for a different John Smith in the company, so must be carefully read, and forwarded just in case. And the unfortunate John Smith who received the email has the problem of figuring out which John Smith he should forward the email to.

One way of solving this problem is that once a company hired someone with a particular name, they just wouldn't hire another. I_2 (with the name Radia Perlman, which is probably unique in the entire world) think that's reasonable, but someone with a name like John Smith might start having problems finding a company that could hire him.

> *Now why did you name your baby* John*? Every Tom, Dick, and Harry is named* John.
>
> —Sam Goldwyn

1.10 AUTHENTICATION AND AUTHORIZATION

Authentication is when Alice proves to Bob that she is Alice. Authorization is having something decide what a requester is allowed to do at service Bob.

1.10.1 ACL (Access Control List)

Typically the way a server decides whether a user should have access to a resource is by first authenticating the user, and then consulting a database associated with the resource that indicates who is allowed to do what with that resource. For instance, the database associated with a file might say that Alice can read it, and George and Carol are allowed to read and write it. This database is often referred to as an **ACL** (**access control list**).

1.10.2 Central Administration/Capabilities

Another model of authorization is that instead of listing, with each resource, the set of authorized users and their rights (*e.g.*, *read*, *write*, *execute*), service Bob might have a database that listed, for each user, everything she was allowed to do. If everything were a single application, then the ACL model and the central administration model would be basically the same, since in both cases there would be a database that listed all the authorized users and what rights each had. But in a world in which there are many resources, not all under control of the same organization, it would be difficult to have a central database listing what each user was allowed to do. This model would have scaling problems if there were many resources each user was allowed to access, especially if resources were created and deleted at a high rate. And if resources are under control of different organizations, there wouldn't be a single organization trusted to manage the authorization information.

Some people worry that ACLs don't scale well if there are many users allowed access to each resource. But the concept of groups helps the scaling issue.

1.10.3 Groups

Suppose there were a file that should be accessible to, say, any Dell employee. It would be tedious to type all the employee names into that file's ACL. And for a set of employees such as "all Dell employees", there would likely be many resources with the same set of authorized users. And it would take a lot of storage to have such huge ACLs on that many resources, and whenever anyone joined or left the company, all those ACLs would have to be modified.

The concept of a group was invented to make ACLs more scalable. It is possible to include a group name on an ACL, which means that any member of the group is allowed access to the resource. Then, group membership can be managed in one place, rather than needing to update ACLs at every resource when the membership changes.

Traditionally, a server that protected a resource with a group named on the ACL needed to know all the members of the group, but if there are many servers that store resources for that group, this would be inefficient and inconvenient. Also, that model would preclude more flexible group mechanisms; for example:

- cross-organizational groups, where no one server is allowed to know all the members;
- anonymous groups, where someone can prove membership in the group without having to divulge their identity.

Traditionally, groups were centrally administered, so it was easy to know all the groups to which a user belonged, and the user would not belong to many groups. But in many situations, it is useful for any user to be able to create a group (such as Alice's friends, or students who have already turned in their exams in my course), and have anyone be able to name such a group on an ACL.

Scaling up this simple concept of users, groups, and ACLs to a distributed environment has not been solved in practice. This section describes various ways that it might be done and challenges with truly general approaches.

1.10.4 Cross-Organizational and Nested Groups

It would be desirable for an ACL to allow any Boolean combination of groups and individuals. For instance, the ACL might be the union of six named individuals and two named groups. If someone is one of the named individuals, or in one of the groups, the ACL would grant them permission. It would also be desirable to have the ACL be something like Group A and NOT group B, *e.g.*, U.S. citizen and not a felon.

Likewise, it would be desirable for group membership to be any Boolean combination of groups and individuals, *e.g.*, the members of Alliance-executives might be CompanyA-execs, CompanyB-execs, and John Smith. Each of the groups Alliance-executives, CompanyA-execs, and CompanyB-execs is likely to be managed by a different organization, and the membership is likely to be stored on different servers. How, then, can a server (Bob) that protects a resource that has the group Alliance-executives on the ACL know whether to allow Alice access? If she's not explicitly listed on the ACL, she might be a member of one of the groups on the ACL. But Bob does not necessarily know all the members of the group. Let's assume that the group name (Alliance-executives) can be looked up in a directory to find out information such as its network address and its public key. Or perhaps the group name would contain the DNS name of the server that manages that group, so the group name might be example.com/Alliance-executives.

- Bob could periodically find every group on any ACL on any resource it protects, and attempt to collect the complete membership. This means looking up all the members of all subgroups, and subgroups of subgroups. This has scaling problems (the group memberships might be very large), performance problems (there might be a lot of traffic with servers querying group membership servers for membership lists), and cache staleness problems. How often would this be done? Once a day is a lot of traffic, but a day is a lot of time to elapse for Alice's group membership to take effect, and for revocations to take effect.
- When Alice requests access, Bob could then ask the on-line group server associated with the group whether Alice is a member of the group. This could also be a performance nightmare with many queries to the group server, especially in the case of unauthorized users creating a denial of service attack by requesting access to services. At the least, once Alice is discovered to either belong or not belong, Bob could cache this information. But again, if the cache is held for a long time, it means that membership can take a long time to take effect, and revocation can also take a long time to take effect.
- When Alice requests access to a resource, Bob could reply, "You are not on the ACL, but here are a bunch of groups that are on the ACL, so if you could prove you are a member of one of those groups, you can access it." Alice could then contact the relevant group servers and attempt to get certification of group membership. Then, she can reply to Bob with some sort of proof that she is a member of one of the groups on the ACL. Again, this could have a cache staleness problem, and a performance problem if many users (including unauthorized troublemakers) contact group servers asking for proof of group membership.

1.10.5 Roles

The term *role* is used in many different ways. Authorization based on roles is referred to as **RBAC** (**role-based access control**). In most usage today, a **role** is the same as what we described as a group. In some cases, Alice will need to log in as the role rather than as the individual. For example, she might log in as `admin`. However, many users might need to log in as `admin`, and for auditing purposes, it is important to know which user was invoking the admin role. Therefore, for auditing purposes, a user might be simultaneously logged in as the role (admin) and as the individual (Alice). In many environments, a role is given a name such as "admin", and being able to invoke the admin role on your own machine should not authorize you to invoke the admin role on controlling a nuclear power plant. So, both groups and roles should have names that specify the domain, *e.g.*, the DNS name of the service that manages the group or role membership.

Although a lot of systems implement roles as the same thing as groups, it is confusing to have two words that mean the same thing. We recommend that the difference between a group and a role is that a role needs to be consciously invoked by a user, often requiring additional authentication such as typing a different password. In contrast, we recommend that with a group, all members automatically have all rights of the group, without even needing to know which groups they are a member of. With roles, users may or may not be allowed to simultaneously act in multiple roles, and perhaps multiple users may or may not be allowed to simultaneously act in a particular role (like *President of the United States*). Again, we are discussing various ways things could work. Deployed systems make different choices. Some things people would like to see roles solve:

- When a user is acting in a particular role, the application presents a different user interface. For instance, when a user is acting as *manager*, the expense reporting utility might present commands for approving expense reports, whereas when the user is acting as *employee*, the application might present commands for reporting expenses.
- Having roles enables a user to be granted a subset of all the permissions they might have. If the user has to explicitly invoke the privileged role, and only keeps themselves authenticated as that role for as long as necessary, it is less likely that a typo will cause them to inadvertently do an undesirable privileged operation.
- Allowing a user to be able to run with a subset of her rights (not invoking her most privileged role except when necessary) gives some protection from malicious code. While running untrusted code, the user should be careful to run in an unprivileged role.
- Sometimes there are complex policies, such as that you are allowed to read either file A or file B but not both. Somehow, proponents of roles claim roles will solve this problem. This sort of policy is called a **Chinese wall**.

1.11 MALWARE: VIRUSES, WORMS, TROJAN HORSES

Lions and tigers and bears, oh my!
—Dorothy (in the movie *The Wizard of Oz*)

People like to categorize different types of malicious software and assign them cute biological terms (if one is inclined to think of worms as cute). We don't think it's terribly important to distinguish between these things, so in the book we'll refer to all kinds of malicious software generically as **malware**. However, here are some of the terms that seem to be infecting the literature.

- **Trojan horse**—instructions hidden inside an otherwise useful program that do bad things. Usually, the term *Trojan horse* is used when the malicious instructions are installed at the time the program is written (and the term *virus* is used if the instructions get added to the program later).
- **virus**—a set of instructions that, when executed, inserts copies of itself into other programs.
- **worm**—a program that replicates itself by installing copies of itself on other machines across a network.
- **trapdoor**—an undocumented entry point intentionally written into a program, often for debugging purposes, which can be exploited as a security flaw. Often people forget to take these out when the product ships. However, sometimes these undocumented "features" are intentionally put into software by an employee who thinks he might at some point become disgruntled.
- **bot**—a machine that has been infected with malicious code that can be activated to do some malicious task by some controller machine across the Internet. The controller is often referred to as a **bot herder**. The intention of the disease-infected rodents (apologies if we offend actual disease-infected rodents who might be reading this book) who turn a computer into a bot is that the owner of the machine should not be aware that their machine is infected. To the owner of the machine, it continues operating as usual. However, the controller can at some point rally all the bots under its control to do some sort of mischief like sending out spam or flooding some service with nuisance messages. The bots under its control are often referred to as a **bot army**. There are price lists on the dark web for renting a bot army, priced according to the size of the army and the amount of time you would like to rent them. (The **dark web** is a subset of the Internet that is not visible to search engines, and requires special access mechanisms. It is an ideal place for purchasing illegal goods.)
- **logic bomb**—malicious instructions that trigger on some event in the future, such as a particular time occurring. The delay might advantage criminals that create a logic bomb, because

they can first deploy the logic bomb in many places. Or, they can make the bad event occur long after they have left the company, so they won't be under suspicion.

- **ransomware**—malicious code that encrypts the user's data, and then the helpful criminal offers to help get the data back, for a small fee (such as several million dollars).

Most of the items above exploit bugs in operating systems or applications. There are also vulnerabilities that exploit bugs in humans. One example is **phishing**, the act of sending email to a huge unstructured list of email addresses, that will trick some small percentage of the recipients into infecting their own machines, or divulging information such as a credit card number. **Spear phishing** means sending custom messages to a more focused group of people.

1.11.1 Where Does Malware Come From?

Where do these nasties come from? Originally, it was hobbyists just experimenting with what they could do. Today, big money can be made with malware. A bot army can be rented to someone that wants to damage a competitor, or a political organization they disagree with. And malware can be used for demanding ransoms. It is also used by sophisticated spy organizations. A notable example is Stuxnet. This was malware aimed specifically at damaging a certain type of equipment (centrifuges) used in Iran, to slow Iran's ability to produce nuclear weapons.

How could an implementer get away with intentionally writing malware into a program? Wouldn't someone notice by looking at the program? A good example of how hard it can be to decipher what a program is doing, even given the source code, is the following nice short program, written by Ian Phillipps (see Figure 1-2).

It was a winner of the 1988 International Obfuscated C Code Contest. It is delightful as a Christmas card. It does nothing other than its intended purpose (I_1 have analyzed the thing carefully and I_2 have complete faith in me_1), but we doubt many people would take the time to understand this program before running it

But also, nobody looks. Often when you buy a program, you do not have access to the source code, and even if you did, you probably wouldn't bother reading it all, or reading it very carefully. Many programs that run have never been reviewed by anybody. A claimed advantage of the "open source" movement (where all software is made available in source code format) is that even if *you* don't review it carefully, there is a better chance that someone else will.

What does a virus look like? A virus can be installed in just about any program by doing the following:

- replace any instruction, say the instruction at location *x*, by a jump to some free place in memory, say location *y*; then
- write the virus program starting at location *y*; then

```
/* Have yourself an obfuscated Christmas! */
#include <stdio.h>
main(t,_,a)
char *a;
{
return!0<t?t<3?main(-79,-13,a+main(-87,1-_,main(-86,0,a+1)+a)):
1,t<_?main(t+1,_,a):3,main(-94,-27+t,a)&&t==2?_<13?
main(2,_+1,"%s %d %d\n"):9:16:t<0?t<-72?main(_,t,
"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l,+,/n{n+,/+#n+,/#\
;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l \
q#'+d'K#!/+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw' i;# \
){nl]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#n'wk nw' \
iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \
;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;#'rdq#w! nr'/ ') }+}{rl#'{n' ')# \
}'+}##(!!/")
:t<-50?_==*a?putchar(31[a]):main(-65,_,a+1):main((*a=='/')+t,_,a+1)
:0<t?main(2,2,"%s"):*a=='/'||main(0,main(-61,*a,
"!ek;dc i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghiry"),a+1);
}
```

Figure 1-2. Christmas Card?

- place the instruction that was originally at location x at the end of the virus program, followed by a jump to $x+1$.

Besides doing whatever damage the virus program does, it might replicate itself by looking for any executable files in any directory and infecting them. Once an infected program is run, the virus is executed again, to do more damage and to replicate itself to more programs. Most viruses spread silently until some triggering event causes them to wake up and do their dastardly deeds. If they did their dastardly deeds all the time, they wouldn't spread as far.

1.11.2 Virus Checkers

How can a program check for viruses? There's rather a race between the brave and gallant people who analyze the viruses and write clever programs to detect and eliminate them, and the foul-smelling scum who devise new types of viruses that will escape detection by all the current virus checkers.

The oldest form of virus checker knows instruction sequences that have appeared in known viruses, but are believed not to occur in benign code. It checks all the files on disk and instructions in memory for those patterns of commands, and raises a warning if it finds a match hidden somewhere inside some file. Once you own such a virus checker, you need to periodically get updates of the patterns file that include the newest viruses.

To evade detection of their viruses, virus creators have devised what are known as a **polymorphic viruses**. When they copy themselves, they change the order of their instructions or change instructions to functionally similar instructions. A polymorphic virus may still be detectable, but it takes more work, and not just a new pattern file. Modern virus checkers don't just periodically scan the disk. They actually hook into the operating system and inspect files before they are written to disk.

Another type of virus checker takes a snapshot of disk storage by recording the information in the directories, such as file lengths. It might even take message digests of the files. It is designed to run, store the information, and then run again at a future time. It will warn you if there are suspicious changes. One virus, wary of changing the length of a file by adding itself to the program, compressed the program so that the infected program would wind up being the same length as the original. When the program was executed, the uncompressed portion containing the virus decompressed the rest of the program, so (other than the virus portion) the program could run normally.

Some viruses attack the virus checkers rather than just trying to elude them. If an attacker can penetrate a company that disseminates code, such as virus signatures or program patches, they can spread their malware to all the customers of that company.

1.12 SECURITY GATEWAY

A security gateway (see Figure 1-3) sits between your internal network and the rest of the network and provides various services. Such a box usually provides many functions, such as firewall (§1.12.1), web proxy (§1.12.2), and network address translation (§1.14). We will describe these features in the next few sections.

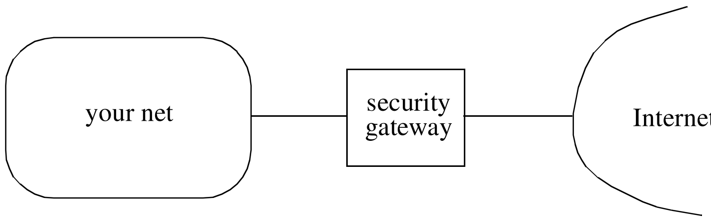

Figure 1-3. Security Gateway

1.12.1 Firewall

There was a time when people assumed that their internal network and all the users and systems on the internal network were trustworthy. The only challenge was connectivity to the Internet. Users need to communicate with publicly available services. The company needs to make some of its own services available to customers located outside the corporate network. So the thought was to install a box between your network and the scary Internet that can keep things secure, somehow.

The belief that a firewall between your internal network and the scary Internet is all you need to protect you has long been discredited. Even if you had the most secure firewall, *i.e.*, a box that blocked any traffic between your network and the Internet, malware inside your network could do arbitrary damage. And even carefully configured firewalls often break legitimate usage.

A recent buzzword is **zero trust**. It basically means the opposite of the previous thinking. Although firewalls can add some amount of extra security, all applications must protect themselves. They must authenticate everything they are talking to and enforce access control rules. The buzzword **defense-in-depth** means having multiple stages of security, so if one fails, hopefully another will protect you. Although people pretty much agree that the old firewall-will-protect-you model is no longer the right way to think about security, a lot of the mechanisms supporting that model are still deployed.

Firewalls centrally manage access to services in ways that individual systems should, but often don't. Firewalls can enforce policies such as *systems outside the firewall can't access file services on any systems inside the firewall*. With such a restriction, even if the internal systems are more open than they should be, or have bugs, they can't be attacked directly from systems outside the firewall.

Note that a firewall need not be a physical box that the company buys. It could instead be software on each machine that does the same sorts of filtering that a firewall box would do. This concept is sometimes called a **distributed firewall**. Both "firewall" and "distributed firewall" are buzzwords, and, as with most buzzwords, the definitions evolve and are used by different vendors in different ways. Usually *distributed firewall* implies that all the components doing "firewall stuff" are centrally managed. Otherwise, it would just be multiple firewalls.

The simplest form of firewall selectively discards packets based on configurable criteria, such as addresses in the IP header, and does not keep state about ongoing connections. For example, it might be configured to only allow some systems on your network to communicate outside, or some addresses outside your network to communicate into your network. For each direction, the firewall might be configured with a set of legal source and destination addresses, and it drops any packets that don't conform. This is known as **address filtering**.

Packet filters usually look at more than the addresses. A typical security policy is that for certain types of traffic (*e.g.*, email, web surfing), the rewards outweigh the risks, so those types of traffic should be allowed through the firewall, whereas other types of traffic (say, remote terminal access), should not be allowed through.

To allow certain types of traffic between host A and B while disallowing others, a firewall can look at the protocol type in the IP header, at the ports in the layer 4 (TCP or UDP) header, and at anything at any fixed offset in the packet. For web traffic, either the source or destination port will likely be 80 (http) or 443 (https). For email, either the source or destination port will likely be 25.

Firewalls can be even fancier. Perhaps the policy is to allow connections initiated by machines inside the firewall, but disallow connections initiated by machines outside the firewall. Suppose machine A inside the firewall initiates a connection to machine B outside the firewall. During the conversation, the firewall will see packets from both directions (A to B as well as B to A), so it can't simply disallow packets from B to A. The way it manages to enforce only connections initiated by A, is to look at the TCP header. TCP has a flag (called ACK) that is set on all but the first packet, the one that establishes the connection. So if the firewall disallows packets from B without ACK set in the TCP header, then it will usually have the desired effect.

Another approach is a **stateful packet filter**, *i.e.*, a packet filter that remembers what has happened in the recent past and changes its filtering rules dynamically as a result. A stateful packet filter could, for instance, note that a connection was initiated from inside using IP address *s*, to IP address *d*, and then allow (for some period of time) connections from IP address *d* to IP address *s*.

1.12.2 Application-Level Gateway/Proxy

An application-level gateway, otherwise known as a **proxy**, acts as an intermediary between a client and the server providing a service. The gateway could have two network adaptors and act as a router, but more often it is placed between two packet-filtering firewalls, using three boxes (see Figure 1-4). The two firewalls are routers that refuse to forward anything unless it is to or from the gateway. Firewall F_2 refuses to forward anything from the global net unless the destination address is the gateway and refuses to forward anything to the global net unless the source is the gateway. Firewall F_1 refuses to forward anything from your network unless the destination address is the gateway, and refuses to forward anything to your network unless the source address is the gateway. To transfer a file from your network to the global network, you could have someone from inside transfer the file to the gateway machine, and then the file is accessible to be read by the outside world. Similarly, to read a file into your network, a user can arrange for it to first get copied to the gateway machine. To log into a machine in the global network, you could first log into the gateway machine, and from there you could access machines in the remote network. An application-level gateway is sometimes known as a **bastion host**. It must be implemented and configured to be very

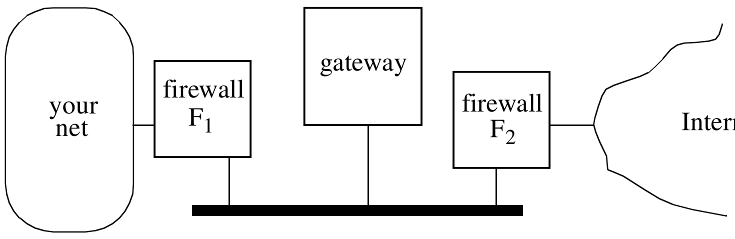

Figure 1-4. Application-Level Gateway

secure. The portion of the network between the two firewalls is known as the **DMZ** (demilitarized zone).

The gateway need not support every possible application. If the gateway does not support an application, either the firewalls on either side of the gateway should block that application (based on layer 4 port), or both firewalls would need to allow that application, depending on whether you want that application to work through the firewall. An example strategy is to allow only electronic mail to pass between your corporate network and the outside world. The intention is to specifically disallow other applications, such as file transfer and remote login. However, electronic mail can certainly be used to transfer files. Sometimes a firewall might specifically disallow very large electronic mail messages, on the theory that this will limit the ability to transfer files. But often, large electronic mail messages are perfectly legitimate, and any file can be broken down into small pieces.

Sometimes an application might specifically be aware of proxies running on an application gateway. For example, browsers can be configured with the address of a proxy, and then all requests will be directed to the proxy. The proxy might be configured with which connections are allowed and which are disallowed, and could even inspect the data passing through to check for malware.

1.12.3 Secure Tunnels

A *tunnel* (see Figure 1-5) is a point-to-point connection created between nodes A and B, where the connection is a path across a network. A and B can treat the tunnel as a direct link between each other. An e*ncrypted tunnel* is when A and B establish a secure session. An encrypted tunnel is sometimes called a **VPN** (virtual private network). We think that's a really bad term, and a more accurate term would be VPL (virtual private link). Whoever decided to call it a VPN imagined that the encrypted tunnel becomes an extra link in your private network, so a network consisting of a combination of private links and encrypted tunnels across the Internet becomes your private network. If we just use the acronym VPN for the encrypted tunnel (like the industry seems to) and don't think about what VPN expands to, I_2 guess we can live with the term.

Suppose the only reason you've hooked into the Internet is to connect disconnected pieces of your own network to each other. Instead of the configuration in Figure 1-5, you could have bought dedicated links between G_1, G_2, and G_3, and trusted those links as part of your corporate network because you owned them. But it's likely to be cheaper to have the Gs pass data across the Internet. How can you trust your corporate data crossing over the Internet? You do this by configuring G_1, G_2, and G_3 with security information about each other (such as cryptographic keys), and creating secure tunnels between them.

The mechanics of the tunnel between G_1 and G_2, from the IP (network layer 3) point of view, is that when A sends a packet to C, A will launch it with an IP header that has source=A, destination=C. When G_1 sends it across the tunnel, it puts it into another envelope, *i.e.*, it adds an

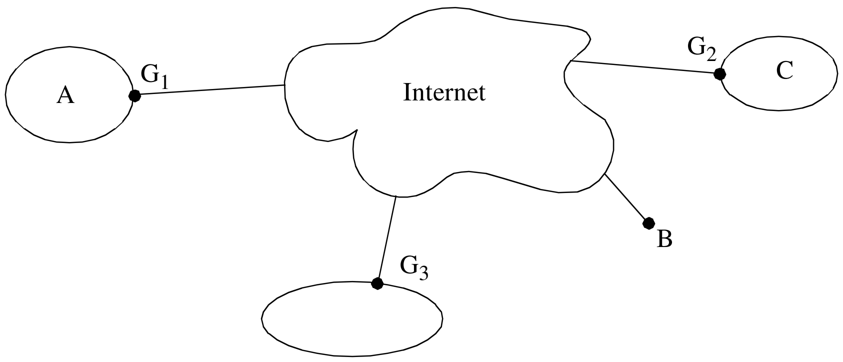

Figure 1-5. Connecting a Private Network over a Public Internet

additional IP header, treating the inner header as data. The outer IP header will contain source=G_1 and destination=G_2. And all the contents (including the inner IP header) will be encrypted and integrity protected, so it is safe to traverse the Internet.

1.12.4 Why Firewalls Don't Work

Firewalls alone (without also doing end-to-end security) assume that all the bad guys are on the outside, and everyone inside can be completely trusted. This is, of course, an unwarranted assumption. Should employees have access (read and write) to the salary database, for instance?

Even if the company is so careful about hiring that no employee would ever do anything bad intentionally, firewalls can be defeated if an attacker can inject malicious code into a machine on the corporate network. This can be done by tricking someone into downloading something from the Internet or launching an executable from an email message. It is quite common for an attacker to break into one system inside your firewall, and then use that system as a platform for attacking other systems. Someone once described firewall-protected networks as "hard and crunchy on the outside; soft and chewy on the inside."

Firewalls often make it difficult for legitimate users to get their work done. The firewall might be configured incorrectly or might not recognize a new legitimate application. And if the firewall allows one application through (say email or http), people figure out how to do what they need to do by disguising it as traffic that the firewall is configured to allow. The ironic term for disguising traffic in order to fool a firewall is to carry your traffic in a **firewall-friendly** protocol. Since firewalls commonly allow http traffic (since it's the protocol used for browsing the web), there are many proposals for doing things over http. The most extreme example is to carry IP over http, which would allow any traffic through! Firewall-friendly? The whole point is to defeat the best efforts of the firewall administrator to disallow what you are doing! It isn't somehow "easier" for the firewall to carry http traffic than any other. The easiest thing for the firewall would be to allow everything or nothing through!

Just as breaking a large file into lots of pieces to be individually carried in separate emails is inefficient, having protocols run on top of *http* rather than simply on top of IP is also inefficient in terms of bandwidth and computation.

1.13 DENIAL-OF-SERVICE (DOS) ATTACKS

A **denial-of-service attack** (**DoS**) is one in which an attacker prevents good guys from accessing a service, but does not enable unauthorized access to any services. In the naive old days, security people dismissed the prospect of denial-of-service attacks as unlikely, since the attacker had nothing to gain. Of course, that turned out to be faulty reasoning. There are terrorists, disgruntled employees, and people who delight in causing mischief for no good reason.

In the earliest types of denial-of-service attacks, the attacker repeatedly sent messages to the victim machine. Most machines at the time were vulnerable to this sort of attack since they had resources that could easily be depleted. For instance, the storage area for keeping track of pending TCP connections tended to be very limited, on the order of, say, ten connections. The probability of ten legitimate users connecting during a single network round trip time was sufficiently small that ten was a reasonable number. But it was easy for the attacking machine to fill up this table on a server, even if the attacking machine was attached to the Internet with a low-speed link.

To avoid being caught at this mischief, it was common for the attacker to send these malicious packets from forged source addresses. This made it difficult to find (and prosecute) the attacker, and it made it difficult to recognize packets from the malicious machine and filter them at a firewall.

As a defense, people advocated having routers have the capability of doing sanity checks on the source address. These routers could be configured to drop packets with a source address that could not have legitimately come from the direction from which the packet was received. Routers might be configured with which source addresses to expect on each of their ports, or they might infer the expected direction from their forwarding tables. This concept was not deployed because it would cause problems. If sanity checks are based on configured information, topological changes in the Internet (such as links going down and alternative routes being used) could cause the routers to make incorrect assumptions. And Mobile IP (RFC 5944) allows a node to move around in the Internet and keep its IP address, which would confuse routers attempting sanity checks.

A deployed defense against a single malicious node attempting to swamp the resources of a server was to increase resources at the server so that a single attacker, at the speeds at which such attackers were typically connected to the Internet, could not fill the pending TCP connection table.

Another level of DoS escalation was to send a single packet that caused a lot of legitimate machines to send messages to the victim machine. An example of such a packet is a packet transmitted to the broadcast address, with the packet's source address forged to the address of the victim's machine, asking for all receivers to respond. All the machines that receive the broadcast will send a response to the victim's machine. Such a mechanism magnifies the effect the attacker can have from his single machine, since each packet he creates turns into *n* packets directed at the victim machine.

As a defense again machines sending packets from forged IP addresses, protocols such as TCP, IPsec, and TLS have been designed to avoid requiring a receiver, Bob, to keep state or do significant computation if requests are arriving from forged IP source addresses. Unless the requester can receive packets at the IP address they claim to be coming from, Bob will not need to keep state about the request. Only when the requester returns something that Bob sent to its claimed IP address will Bob pay attention to this request.

But then came the next level of escalation, which is known as a **distributed-denial-of-service attack** (**DDoS**). In this form of attack, the attacker breaks into a lot of innocent machines, and installs software on them to have them all attack the victim machine. These innocent machines are called **zombies** or **drones** or **bots**. With enough bots attacking it, any machine can be made inaccessible, since even if the machine itself can process packets as fast as they can possibly arrive, the links or routers in front of that machine can be overwhelmed. The defenses in TCP, IPsec, and TLS will not help, since the bot machines are using their own IP addresses in the requests. Since requests are coming from hundreds or thousands of innocent machines, it is hard to distinguish these packets from packets coming from legitimate users.

1.14 NAT (NETWORK ADDRESS TRANSLATION)

NAT was designed, out of necessity, because IPv4 addresses were too small (four bytes) to give unique addresses to every node on the Internet. With NAT, a piece of the Internet (say a corporate network) can use IP addresses that are not globally unique, and, in fact, these addresses are reused in many other networks. This means that a node inside such a network cannot be contacted from outside that network. However, if the security gateway provides NAT functionality, it will have a pool of globally unique IP addresses that can be assigned as needed.

The NAT box will almost certainly not have a large enough pool of IP addresses to give a globally reachable IP address to every internal node communicating to outside nodes. So the NAT box also translates the TCP or UDP port as well. So a NAT implementation might have a mapping of ⟨internal IP, internal port⟩ maps to ⟨external IP, external port⟩. When internal node Alice sends a packet to external destination Bob, the NAT box will replace the source IP and port to the external

IP and port in the NAT box's table. Likewise, when packets arrive from Bob for Alice's assigned external ⟨IP, port⟩, the NAT box replaces these fields in the destination fields in Bob's packet before forwarding the packet on the internal network.

There is somewhat of a security problem with this approach if the NAT box simply translates tuples of ⟨IP address, port⟩. Suppose Alice starts a connection to Bob, and the NAT box then creates an entry to translate Alice's internal address and port to, for instance, globally reachable tuple ⟨ IP_{Alice}, $Port_{Alice}$⟩. If the NAT box will forward anything to Alice that is addressed to Alice's temporarilty assigned global address and port, then any node on the Internet could send a packet to Alice by addressing it to ⟨IP_{Alice}, $Port_{Alice}$⟩. Sometimes this is the desired behavior. For example, assume there is a conferencing system. Alice, George, and Carol (all behind NATs) join the conference by contacting the central server, but it is not desirable for all of the conference communication to go through the central server. If the NAT box allows anyone that knows Alice's temporary global address to contact Alice, then the conference coordinator can tell all of the members the other members' global addresses, and they can then directly communicate with each other.

To create the behavior that only the node that Alice has initiated a connection to, to be able to reach Alice, then the NAT box will keep a mapping of 4-tuples to ⟨external IP, port⟩ pairs. For example, if Alice, at internal IP=a, internal port=p initiates a connection to external Bob, at IP address=B, port=P_B, the NAT box might assign Alice, for this connection, the external address and port ⟨IP_A, $Port_A$⟩. The NAT table would include Bob's address in the mapping, and only allow packets from Bob's address and port to be forwarded to Alice. So the NAT entry would have a six-tuple ⟨B, P_B, IP_A, $Port_A$, a, p⟩, meaning that only packets from Bob (at ⟨B, P_B⟩) would be translated and forwarded to Alice.

Another fortuitous use of NATs is for all the devices inside a home to have the same IP address. This is useful because some ISPs (Internet Service Providers) charged for Internet connectivity per device, and the NAT box made it appear to the ISP as if there were only a single device in the house. The ISP's answer to this threat to their pricing model was to include in the 74-page EULA (end user license agreement) that everyone has to click on (but nobody reads), an agreement by the user not to use a NAT box. Now, if an actual human read the 74-page agreement, they would most likely think "What's a NAT box?"

1.14.1 Summary

These are the main concepts in the Internet. We will give more details about these as they come up in the book.

The Internet has evolved from the original design, and the design was never the only or best way to design a network, but the industry has made it work. An analogy is the English language. It might be overly complicated, with all the spelling and grammar exceptions, but it does the job. And each year some mysterious panel of people decide which new words should officially be added to English, and how the grammar rules should change.

Similarly, for the Internet. If there is anything it can’t do, at least so far, the world has figured out how to evolve the Internet to do what is needed.

This page intentionally left blank

INDEX