Foundations of Security

What Every Programmer Needs to Know

Neil Daswani, Christoph Kern,
and Anita Kesavan

Apress®

Foundations of Security: What Every Programmer Needs to Know

Copyright © 2007 by Neil Daswani, Christoph Kern, and Anita Kesavan

ISBN-13 (pbk): 978-1-59059-784-2

ISBN-10 (pbk): 1-59059-784-2

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Jonathan Gennick
Technical Reviewer: Dan Pilone
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Matt Wade
Project Manager: Kylie Johnston
Copy Edit Manager: Nicole Flores
Copy Editor: Damon Larson
Assistant Production Director: Kari Brooks-Copony
Production Editor: Ellie Fountain
Compositor: Dina Quan
Proofreader: Liz Welch
Indexer: Julie Grady
Artist: Kinetic Publishing Services, LLC
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit http://www.springeronline.com.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com in the Source Code/Download section.

This book is dedicated to Dad, who provided me my foundations,
and Mom, who taught me what I needed to know.
—N. Daswani

Contents at a Glance

Contents

Foreword

When Neil Daswani and Christoph Kern invited me to write a foreword to the book you are reading now, I accepted without hesitation and with a good deal of pleasure. This timely volume is solidly grounded in theory and practice and is targeted at helping programmers increase the security of the software they write. Despite the long history of programming, it seems as if bug-free and resilient software continues to elude us. This problem is exacerbated in networked environments because attacks against the vulnerabilities in software can come from any number of other computers and, in the Internet, that might mean millions of potential attackers. Indeed, the computers of the Internet that interact with each other are in some sense performing unplanned and unpredictable tests between the software complements of pairs of machines. Two machines that start out identically configured will soon become divergent as new software is downloaded as a consequence of surfing the World Wide Web or as updates are applied unevenly among the interacting machines. This richly diverse environment exposes unexpected vulnerabilities, some of which may be exploited deliberately by hackers intent on causing trouble or damage and who may even have pecuniary motivations for their behavior. So-called bot armies are available in the millions to be directed against chosen targets, overwhelming the defenses of some systems by the sheer volume of the attack. In other cases, known weaknesses are exploited to gain control of the target machines or to introduce viruses, worms, or Trojan horses that will do further damage.

Programmers writing for networked environments have a particularly heavy responsibility to be fully aware of the way in which these vulnerabilities may come about and have a duty to do everything they can to discover and remove them or to assure that they are eliminated by careful design, implementation, and testing. It takes discipline and a certain amount of paranoia to write secure software. In some ways it is like driving defensively. You must assume you are operating in a hostile environment where no other computer can be trusted without demonstrating appropriate and verifiable credentials. Even this is not enough. In a kind of nightmare scenario, someone with a USB memory stick can bypass all network defenses and inject software directly into the computer. Such memory sticks emulate disks and can easily pick up viruses or worms when they are used on unprotected computers, and when reused elsewhere, can propagate the problem. All input must be viewed with suspicion until cleared of the possibility of malformation.

Vulnerability can exist at all layers of the Internet protocol architecture and within the operating systems. It is naive to imagine that simply encrypting traffic flowing between pairs of computers on the Internet is sufficient to protect against exploitation. An obvious example is a virus attached to an e-mail that is sent through the Internet fully encrypted at the IP layer using IPsec. Once the message is decrypted packet by packet and reassembled, the virus will be fully ready to do its damage unless it is detected at the application layer by the e-mail client, or possibly by the mail transport agent that delivers the e-mail to the target recipient.

It is vital to understand not only how various attacks are carried out, but also how the vulnerabilities that enable these attacks arise. Programs that fail to check that inputs are properly sized or have appropriate values may be vulnerable to buffer overruns leading to application or even operating system compromise. Failure to verify that input is coming in response to a request could lead to database pollution (this is one way that Domain Name System resolvers can end up with a "poisoned" cache). Among the most pernicious of network-based attacks are the denial-of-service attacks and the replay attacks that resend legitimately formatted information at the target in the hope of causing confusion and malfunction.

In this book, Daswani and Kern have drawn on real software vulnerabilities and network-based threats to provide programmers with practical guidelines for defensive programming. Much of the material in this book has been refined by its use in classroom settings with real programmers working on real problems. In the pursuit of security, there is no substitute for experience with real-world problems and examples. Abstracting from these concrete examples, the authors develop principles that can guide the design and implementation and testing of software intended to be well protected and resilient against a wide range of attacks.

Security is not only a matter of resisting attack. It is also a matter of designing for resilience in the face of various kinds of failure. Unreliable software is just as bad as software that is vulnerable to attack, and perhaps is worse because it may fail simply while operating in a benign but failure-prone setting. Fully secure software is therefore also designed to anticipate various kinds of hardware and software failure and to be prepared with remediating reactions. Good contingency planning is reliant on imagination and an ability to compose scenarios, however unlikely, that would render false the set of assumptions that might guide design for the "normal" case. The ability to anticipate the possible, if unlikely, situations—the so-called "corner" cases—is key to designing and implementing seriously resilient software.

It is a pleasure to commend this book to your attention. I share with its authors the hope that it will assist you in the production of increasingly secure and resilient software upon which others may rely with confidence.

Vinton G. Cerf
Vice President and Chief Internet Evangelist, Google

About the Authors

NEIL DASWANI, PHD, has served in a variety of research, development, teaching, and managerial roles at Google, NTT DoCoMo USA Labs, Stanford University, Yodlee, and Telcordia Technologies (formerly Bellcore). While at Stanford, Neil cofounded the Stanford Center Professional Development (SCPD) Security Certification Program. His areas of expertise include security, peer-to-peer systems, and wireless data technology. He has published extensively in these areas, he frequently gives talks at industry and academic conferences, and he has been granted several US patents. He received a PhD in computer science from Stanford University. He also holds an MS in computer science from Stanford University, and a BS in computer science with honors with distinction from Columbia University.

CHRISTOPH KERN is an information security engineer at Google, and was previously a senior security architect at Yodlee, a provider of technology solutions to the financial services industry. He has extensive experience in performing security design reviews and code audits, designing and developing secure applications, and helping product managers and software engineers effectively mitigate security risks in their software products.

ANITA KESAVAN is a freelance writer and received her MFA in creative writing from Sarah Lawrence College. She also holds a BA in English from Illinois-Wesleyan University. One of her specializations is communicating complex technical ideas in simple, easy-to-understand language.

About the Technical Reviewer

DAN PILONE is a senior software architect with Pearson Blueprint Technologies and the author of *UML 2.0 in a Nutshell*. He has designed and implemented systems for NASA, the Naval Research Laboratory, and UPS, and has taught software engineering, project management, and software design at the Catholic University in Washington, DC.

Acknowledgments

There are many who deserve much thanks in the making of this book. Firstly, I thank God for giving me every moment that I have on this earth, and the opportunity to make a positive contribution to the world.

I thank my wife, Bharti Daswani, for her continuous patience, understanding, and support throughout the writing process; for her great attitude; and for keeping my life as sane and balanced as possible while I worked to finish this book. I also thank her for her forgiveness for all the nights and weekends that I had to spend with "the other woman"—that is, my computer! I'd also like to thank my parents and my brother for all the support that they have provided over the years. Without my parents' strong focus on education and my brother's strong focus on giving me some competition, I am not quite sure I would have been as driven.

I'd like to thank Gary Cornell for taking on this book at Apress. I remember reading Gary's *Core Java* book years ago, and how much I enjoyed meeting him at the JavaOne conference in 2001. Even at that time, I was interested in working on a book, but I had to complete my PhD dissertation first! I'd like to thank my editor, Jonathan Gennick, for always being so reasonable in all the decisions that we had to make to produce this book. I'd like to thank Dan Pilone, my technical reviewer from Apress, for his critical eye and for challenging many of my examples. I thank Kylie Johnston for bearing with me as I juggled many things while concurrently writing this book; she was instrumental to keeping the project on track. Damon Larson and Ellie Fountain deserve thanks for their diligent contributions toward the copy editing and production preparation for the book.

This book has benefited from a distinguished cast of technical reviewers. I am grateful for all their help in reviewing various sections of this book. The book has benefited greatly from all their input—any mistakes or errors that remain are my own fault. The technical reviewers include Marius Schilder, Alma Whitten, Heather Adkins, Shuman Ghosemajumder, Kamini Mankaney, Xavier Pi, Morris Hoodye, David Herst, Bobby Holley, and David Turner.

I'd like to thank Vint Cerf for serving as an inspiration, and for taking the time to meet with me when I joined Google. I didn't mind having to defend my dissertation all over again in our first meeting! I'd like to thank Gary McGraw for introducing me to the field of software security and providing a technical review of this book. I thank Amit Patel for his technical review of the book, and for trading stories about interesting security vulnerabilities that we have seen over the years.

I'd like to thank Dan Boneh for giving me my first project in the field of security, and for the experience of burning the midnight oil developing digital cash protocols for Palm Pilots early in my career at Stanford. I thank Hector Garcia-Molina, my dissertation advisor, for teaching me to write and reason, and about many of the nontechnical things that a scientist/ engineer needs to know.

I thank my coauthors, Christoph Kern and Anita Kesavan, for making the contributions that they have to this book. Without Christoph's in-depth reviews and contributed chapters, this book would have probably had many more errors, and could not provide our readers with

as much depth in the area of cross-domain attacks and command injection. Without Anita's initial help in transcription, editing, and proofreading, I probably would not have decided to write this book.

I'd like to thank Arkajit Dey for helping proofread, edit, and convert this book into different word processing formats. I was even glad to see him find errors in my code! Arkajit is a prodigy in the making, and we should expect great things from him.

We would like to thank Filipe Almeida and Alex Stamos for many fruitful discussions on the finer points of cross-domain and browser security. The chapter on cross-domain security would be incomplete without the benefit of their insights.

I would like to thank Larry Page and Sergey Brin for showing the world just how much two graduate students can help change the world. Their focus on building an engineering organization that is fun to work in, and that produces so much innovation, reminds me that the person who invented the wheel was probably first and foremost an engineer, and only second a businessperson.

Finally, I thank my readers who have gotten this far in reading this acknowledgments section, as it was written on a flight over a glass of wine or two!

Neil Daswani, Ph.D.

Preface

Dr. Gary McGraw, a well-known software security expert, said, "First things first—make sure you know how to code, and have been doing so for years. It is better to be a developer (and architect) and then learn about security than to be a security guy and try to learn to code" (McGraw 2004). If you are interested in becoming a security expert, I wholeheartedly agree with him. At the same time, many programmers who just need to get their job done and do not necessarily intend to become security experts also do not necessarily have the luxury of pursuing things in that order. Often, programmers early in their careers are given the responsibility of producing code that is used to conduct real business on the Web, and need to learn security while they are continuing to gain experience with programming. This book is for those programmers—those who may have (at most) just a few years of experience programming. This book makes few assumptions about your background, and does its best to explain as much as it can. It is not necessarily for people who want to become security experts for a living, but it instead helps give a basic introduction to the field with a focus on the essentials of *what every programmer needs to know about security.*

One might argue that our approach is dangerous, and that we should not attempt to teach programmers about security until they are "mature" enough. One might argue that if they do not know everything they need to know about programming before they learn about security, they might unknowingly write more security vulnerabilities into their code. We argue that if we do not teach programmers *something* about security, they are going to write vulnerabilities into their code anyway! The hope is that if we teach programmers something about security early in their careers, they will probably write fewer vulnerabilities into their code than they would have otherwise, and they may even develop a "spidey sense" about when to ask security professionals for help instead of writing code in blissful ignorance about security.

That said, the goal of this book is to provide enough background for you to develop a good intuition about what might and might not be secure. We do not attempt to cover every possible software vulnerability in this book. Instead, we sample some of the most frequent types of vulnerabilities seen in the wild, and leave it to you to develop a good intuition about how to write secure code. After all, new types of vulnerabilities are identified every day, and new types of attacks surface every day. Our goal is to arm you with principles about how to reason about threats to your software, give you knowledge about how to use some basic defense mechanisms, and tell you where you can go to learn more. (Hence, we have included many references.)

Chief information and security officers can use this book as a tool to help educate software professionals in their organizations to have the appropriate mindset to write secure software. This book takes a step toward training both existing and new software professionals on how to build secure software systems and alleviate some of the common vulnerabilities that make today's systems so susceptible to attack.

Software has become part of the world's critical infrastructure. We are just as dependent upon software as we are on electricity, running water, and automobiles. Yet, software engineering has not kept up and matured as a field in making sure that the software that we rely on is safe and secure. In addition to the voluminous amount of bad press that security vulnerabilities have generated for software companies, preliminary security economics research indicates that a public software company's valuation drops after the announcement of each vulnerability (Telang and Wattal 2005).

Most students who receive degrees in computer science are not required to take a course in computer security. In computer science, the focal criteria in design have been correctness, performance, functionality, and sometimes scalability. Security has not been a key design criterion. As a result, students graduate, join companies, and build software and systems that end up being compromised—the software finds its way to the front page of press articles on a weekly (or daily) basis, customers' personal information that was not adequately protected by the software finds its way into the hands of criminals, and companies lose the confidence of their customers.

The rampant spread of computer viruses and overly frequent news about some new variant worm or denial-of-service attack are constant reminders that the field has put functionality before security and safety. Every other major field of engineering ranging from civil engineering to automobile engineering has developed and deployed technical mechanisms to ensure an appropriate level of safety and security. Every structural engineer learns about the failures of the Tacoma Narrows bridge.[1] Automobile engineers, even the ones designing the cup holders in our cars, think about the safety and security of the car's passengers—if the car ends up in an accident, can the cup holder break in a way that it might stab a passenger?

Unfortunately, it might be hard to argue that the same level of rigor for safety and security is taught to budding software engineers. Safety and security have taken precedence in other engineering fields partially because students are educated about them early in their careers. The current situation is untenable—today's software architects, developers, engineers, and programmers need to develop secure software from the ground up so that attacks can be prevented, detected, and contained in an efficient fashion. Computer security breaches are expensive to clean up after they have happened. Corporate firewalls are often just "turtle shells" on top of inherently insecure systems, and in general are not enough to prevent many types of attacks. Some simple attacks might bounce off the shell, but a hacker just needs to find one soft spot to cause significant damage. Most of these attacks can be stopped.

To complement other software security books that focus on a broader or narrower a range of security vulnerabilities, this book closely examines the 20 percent of the types of vulnerabilities that programmers need to know to mitigate 80 percent of attacks. Also, while this book does not focus on various tips and tricks that might encourage a "band-aid" approach to security, it does teach you about security goals and design principles, illustrates them through many code examples, and provides general techniques that can be used to mitigate large classes of security problems.

Our focus on teaching you how to have a paranoid mindset will also allow you to apply the design principles and techniques we cover to your particular programming tasks and challenges, irrespective of which programming languages, operating systems, and software

1. The original Tacoma Narrows bridge was a suspension bridge built in 1940 in Washington State that employed plate girders to support the roadbed instead of open lattice beam trusses. The bridge violently collapsed four months after its construction due to a 42-mile-per-hour wind that induced a twisting motion that was not considered when the bridge was first designed. The structural collapse was captured on video (see `www.archive.org/details/Pa2096Tacoma`), and is still discussed to this day in many introductory structural and civil engineering classes.

environments you use. Unlike most software books, which are dry and filled with complex technical jargon, this book is written in a simple, straightforward fashion that is easy to read and understand. This book contains many, many examples that allow you to get a deeper practical understanding of computer security. In addition, we use a running example analyzing the security of a functional web server to illustrate many of the security design principles we discuss.

This book is based on the tried-and-tested curriculum for the Stanford Center for Professional Development (SCPD) Computer Security Certification (see `http://proed.stanford.edu/?security`). Many companies and software professionals have already benefited from our course curriculum, and we hope and expect that many more will benefit from this book.

Who This Book Is For

This book is written for programmers. Whether you are studying to be a programmer, have been a programmer for some time, or were a programmer at some point in the past, this book is for you. This book may also be particularly interesting for web programmers, as many of the examples are drawn from the world of web servers and web browsers, key technologies that have and will continue to change the world in ways that we cannot necessarily imagine ahead of time.

For those who are studying to be programmers, this book starts with teaching you the principles that you need to know to write your code in a paranoid fashion, and sensitizes you to some of the ways that your software can be abused by evil hackers once it has been deployed in the real world. The book assumes little about your programming background, and contains lots of explanations, examples, and references to where you can learn more.

This book is also written to be read by those who have been programming for some time, but, say, have never been required to take a course in security. (At the time of writing of this book, that probably includes more than 90 percent of the computer science graduates in the world.) It is written so that it can be the first book you read about computer security, but due to its focus on what security should mean for application programmers (as opposed to system administrators), it will help you significantly build on any existing knowledge that you have about network or operating systems security.

Finally, if you used to be a programmer (and are now, say, a product manager, project manager, other type of manager, or even the CIO/CSO of your company), this book tells you what you need to do to instill security in your products and projects. I'd encourage you to share the knowledge in this book with the programmers that you work with. For those of you who are CIOs or CSOs of your company, this book has been written to serve as a tool that you can provide to the programmers in your company so that they can help you mitigate risk due to software vulnerabilities.

How This Book Is Structured

This book is divided into three parts, and has exercises at the end of each of the parts. The first part focuses on what your goals should be in designing secure systems, some high-level approaches and methodologies that you should consider, and the principles that you should employ to achieve security.

The second part starts with a chapter that covers worms and other malware that has been seen on the Internet. The chapter is meant to scare you into understanding how imperative security is to the future of the entire Internet. While many feel that the topic of worms may be sufficiently addressed at the time of writing of this book, I am not quite sure that I see any inherent reason that the threat could not return in full force if we make mistakes in designing and deploying the next generation of operating system, middleware, and applications software. The chapters following that discuss particular types of vulnerabilities that have caused much pain, such as buffer overflows and additional types of vulnerabilities that have sprung up over the past few years (including client-state manipulation, secure session management, command injection, and cross-domain attacks). In the second part of the book, we also include a chapter on password management, as the widespread use of passwords coupled with badly designed password management systems leads to easily exploitable systems.

The third part of the book provides you with an introduction to cryptography. Cryptography can be an effective tool when used correctly, and when used under the advice and consultation of security experts. The chapters on cryptography have been provided to give you a fluency with various techniques that you can use to help secure your software. After you read the cryptography chapters in this book, if you feel that some of the techniques can help your software achieve its security goals, you should have your software designs and code reviewed by a security expert. This book tells you what you need to know about security to make sure you don't make some of the most common mistakes, but it will not make you a security expert—for that, years of experience as well as additional instruction will be required. At the same time, reading this book is a great first step to learning more about security.

In addition to reading the chapters in this book, we strongly encourage you to do the exercises that appear at the end of each part. Some of the exercises ask concept-based questions that test your understanding of what you have read, while others are hands-on programming exercises that involve constructing attacks and writing code that defends against them. In the world of security, the devil is often in the details, and doing the exercises will give you a much deeper, more detailed understanding to complement your readings. Doing these exercises will help you to walk the walk—not just talk the talk.

If you are an instructor of a computer security course, have the students read the first three chapters and do the exercises. Even if you don't have them do all the exercises at the end of each part of the book, or if you perhaps provide your own complementary exercises, I would recommend that at least some of the exercises that you give them be programming exercises. Chapter 5 could be considered optional, as it is meant to provide some history—at the same time, learning history helps you prevent repeating mistakes of the past. This book is meant to be read from cover to cover, and I believe it holds true to its title in that every programmer should know all of the material in this book, especially if they will be writing code that runs on the Web and handles real user data.

To help those of you who will be teaching security courses, we provide slides based on the material in this book for free at `www.learnsecurity.com`. Each slide deck corresponds to a chapter, and illustrates the same examples that are used in the text, such that the students' readings can reinforce the material discussed in lectures. If you choose to use this book as a required or optional text for your course, the slides can help you save time so that you can focus on the delivery of your course. If your institution has decided to beef up its security training, and you need to get yourself trained so that you can teach the students, I would highly recommend completing both the Fundamental and Advanced Security Certifications at the Stanford Center for Professional Development. There are also many other security training programs in the market, and you are free to choose from any of them. However, due to the young state that the field is in, I would encourage you to choose cautiously, and understand the goals of any particular training program. Is the goal to simply give students a label that they can put on their résumés, or does the program have enough depth to enable to students to solve the real, underlying software security problems that an organization faces?

Conventions

In many parts of this book, we use URLs to refer to other works. Such practice is sometimes criticized because the Web changes so rapidly. Over time, some of the URLs that this book refers to will no longer work. However, as that happens, we encourage readers to use Internet archive-like services such as the Wayback Machine at `www.archive.org` to retrieve old versions of documents at these URLs when necessary. Now, let's just hope that the Wayback Machine and/or other Internet archives continue to work!

Although we may refer to UNIX or Linux in various parts of the text, comments that we make regarding them generally hold true for various flavors of UNIX-based operating systems.

This book has a lot of information, and some of the content has subtleties. We try to point out some of the subtleties in many cases in footnotes. I would recommend reading the footnotes the second time around so that you don't get distracted during your first read through this book.

Prerequisites

This book has no prerequisites, except that you have an interest in programming and security, and have perhaps done some small amount of programming in some language already.

Downloading the Code

All the code examples in this book are available at `www.learnsecurity.com/ntk`, as well as in ZIP file format in the Source Code/Download section of the Apress web site.

Contacting the Authors

Neil Daswani can be contacted at `www.neildaswani.com` and `daswani@learnsecurity.com`.

Christoph Kern can be contacted at `xtof@xtof.org`.

Anita Kesavan can be contacted at `www.anitakesavan.com` and `anita.kesavan@gmail.com`.

PART 1

Security Design Principles

CHAPTER 1

Security Goals

The two main objectives in the first three chapters of this book are to establish the key goals of computer security and to provide an overview of the core principles of secure systems design.

Chapter 1 focuses on the role that technological security plays in the design of a large secure system, and the seven key concepts in the field of security:

- Authentication
- Authorization
- Confidentiality
- Data/message integrity
- Accountability
- Availability
- Non-repudiation

After discussing these concepts, we will then illustrate the role they play in the scope of a larger system by looking at an example of a web client interacting with a web server, and examining the contribution these key concepts make in that interaction.

1.1. Security Is Holistic

Technological security and all the other computer security mechanisms we discuss in this book make up only one component of ensuring overall, holistic security to your system. By technological security, we mean application security, operating system (OS) security, and network security. In addition to discussing what it means to have application, OS, and network security, we will touch upon physical security, and policies and procedures. Achieving holistic security requires physical security, technological security, and good policies and procedures. Having just one or two of these types of security is usually not sufficient to achieve security: all three are typically required. An organization that has advanced technological security mechanisms in place but does not train its employees to safeguard their passwords with care will not be secure overall. The bulk of this book focuses on technological security, and we briefly comment on physical security and policies and procedures in this chapter, as security is holistic. However, our coverage of physical security and policies and procedures do not do the topics justice—for more information, we would encourage you to do the following:

- Read standards such as ISO 17799 (see `www.iso.org/iso/en/prods-services/popstds/informationsecurity.html` and `www.computersecuritynow.com`).
- Visit sites such as the SANS Security Policy Project (`www.sans.org/resources/policies`) for more information on security policies and procedures.
- Read *Practical UNIX and Network Security*, by Simson Garfinkel, Gene Spafford, and Alan Schwartz, for more on operating system and network security.

1.1.1. Physical Security

Physically securing your system and laying down good policies for employees and users is often just as important as using the technological security mechanisms that we cover in this book. All of your servers should be behind locked doors, and only a privileged set of employees (typically system and security administrators) should have access to them. In addition, data centers used to house farms of servers can employ cameras, card reader and biometric locks, and even "vaults" of various kinds, depending upon the sensitivity of data stored on the servers.

In addition to mechanisms that limit access to a physical space to prevent asset theft and unauthorized entry, there are also mechanisms that protect against information leakage and document theft. Documents containing sensitive information can be shredded before they're disposed of so that determined hackers can be prevented from gathering sensitive information by sifting through the company's garbage. Such an attack is often referred to as *dumpster diving*.

1.1.2. Technological Security

In addition to physical security, there are many technical levels of security that are important. Technological security can be divided into three components: application security, OS security, and network security.

Note that our use of the word *technological* to group together application, OS, and network security may not be the best of terms! Clearly, various types of technology can also be used to achieve physical security. For example, employees can be given electronic badges, and badge readers can be put on the door of the server room. The badges and readers clearly employ technology—but here, we use the term *technological security* to refer to software-related application, OS, and network security technology.

Application Security

A web server is an example of an application that can suffer from security problems. In this chapter, and throughout the rest of the book, we use web servers to illustrate many application-layer security vulnerabilities. The deployment scenario that we have in mind for a web server is shown in Figure 1-1.

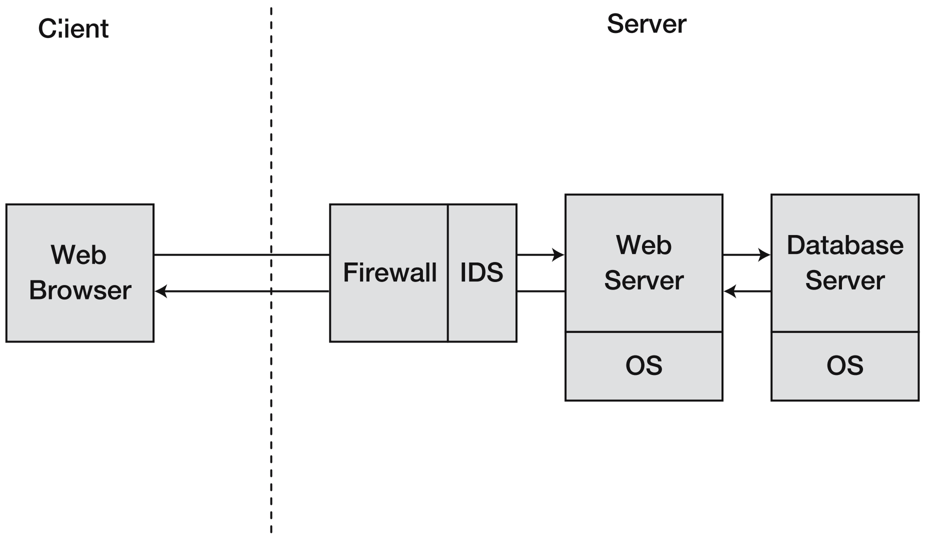

Figure 1-1. *A typical web server deployment scenario*

Consider a scenario in which a web server is configured to allow only certain users to download valuable documents. In this scenario, a vulnerability can arise if there is a bug in how it ascertains the identity of the user. If the user's identity is not ascertained properly, it may be possible for an attacker to get access to valuable documents to which she would not otherwise be granted access.

In addition to ensuring that there are no flaws in the identity verification process used by a web server, it is also important that you configure your web server correctly. Web servers are complicated pieces of software that have many options that can be turned on or off. For instance, a web server may have an option that, when turned on, allows it to serve content from a database; or when turned off, only allows it to serve files from its local file system. Administrators must ensure that their web servers are configured correctly, so as to minimize the possible methods of attack.

By restricting a web server to only serve files from its local file system, you prevent an attacker from taking advantage of vulnerabilities in how a web server uses a back-end database. It is possible for a malicious user to trick a web server into sending user-submitted data to the database as a command, and thereby take control of the database. (One example of such an attack is a SQL injection attack—we cover such attacks in Chapter 8.)

However, even if a web server is not configured to connect to a database, other configuration options might, for instance, make available files on the local file system that a web server administrator did not intend to make accessible. For instance, if a web server is configured to make all types of files stored on its file system available for download, then any sensitive spreadsheets stored on the local file system, for example, could be downloaded just as easily as web documents and images. An attacker may not even need to probe web servers individually to find such documents. A search engine can inadvertently crawl and index sensitive documents, and the attacker can simply enter the right keywords into the search engine to discover such sensitive documents (Long 2004).

Another example of an application that could have a security vulnerability is a web browser. Web browsers download and interpret data from web sites on the Internet. Sometimes web browsers do not interpret data in a robust fashion, and can be directed to download data from malicious web sites. A malicious web site can make available a file that exploits a vulnerability in web browser code that can give the attacker control of the machine that the web browser is running on. As a result of poor coding, web browser code needs to be regularly "patched" to eliminate such vulnerabilities, such as buffer overflows (as discussed in Chapter 6). The creators of the web browser can issue patches that can be installed to eliminate the vulnerabilities in the web browser. A *patch* is an updated version of the software. The patch does not have to consist of an entirely updated version, but may contain only components that have been fixed to eliminate security-related bugs.

OS Security

In addition to application security, OS security is also important. Your operating system—whether it is Linux, Windows, or something else—also must be secured. Operating systems themselves are not inherently secure or insecure. Operating systems are made up of tens or hundreds of millions of lines of source code, which most likely contain vulnerabilities. Just as is the case for applications, OS vendors typically issue patches regularly to eliminate such vulnerabilities. If you use Windows, chances are that you have patched your operating system at least once using the Windows Update feature. The Windows Update feature periodically contacts Microsoft's web site to see if any critical system patches (including security patches) need to be installed on your machine. If so, Windows pops up a small dialog box asking you if it is OK to download the patch and reboot your machine to install it.

It is possible that an attacker might try to exploit some vulnerability in the operating system, even if you have a secure web server running. If there is a security vulnerability in the operating system, it is possible for an attacker to work around a secure web server, since web servers rely on the operating system for many functions.

Network Security

Network layer security is important as well—you need to ensure that only valid data packets are delivered to your web server from the network, and that no malicious traffic can get routed to your applications or operating system. Malicious traffic typically consists of data packets that contain byte sequences that, when interpreted by software, will produce a result unexpected to the user, and may cause the user's machine to fail, malfunction, or provide access to privileged information to an attacker. Firewalls and intrusion detection systems (IDSs) are two types of tools that you can use to help deal with potentially malicious network traffic.

1.1.3. Policies and Procedures

Finally, it is important to recognize that even if your system is physically and technologically secure, you still need to establish a certain set of policies and procedures for all of your employees to ensure overall security. For example, each employee may need to be educated to never give out his or her password for any corporate system, even if asked by a security administrator. Most good password systems are designed so that security and system administrators have the capability to reset passwords, and should never need to ask a user for her existing password to reset it to a new one.

Attackers can potentially exploit a gullible employee by impersonating another employee within the company and convincing him (say, over the phone) to tell him his username or password. Such an attack is called a *social engineering* attack, and is geared at taking advantage of unsuspecting employees. Even if your applications, operating systems, and networks are secure, and your servers are behind locked doors, an attacker can still conduct social engineering attacks to work around the security measures you have in place.

As evidenced by the threat of social engineering, it is important to have policies and procedures in place to help guard sensitive corporate information. Writing down such policies and procedures on paper or posting them on the company intranet is not enough. Your employees need to be aware of them, and they need to be educated to be somewhat paranoid and vigilant to create a secure environment. A combination of physical security, technological security mechanisms, and employees who follow policies and procedures can result in improved overall security for your environment.

It is often said that "security is a process, not a product" (Schneier 2000). There is much more to security than just technology, and it is important to weigh and consider risks from all relevant threat sources.

ARCHETYPAL CHARACTERS

We are going to spend the rest of this chapter illustrating seven key technological security goals (authentication, authorization, confidentiality, message/data integrity, accountability, availability, and non-repudiation). We will do so with the help of a few fictitious characters that are often used in the field of computer security. The first two fictitious characters are Alice and Bob, who are both "good guys" trying to get some useful work done. Their work may often involve the exchange of secret information. Alice and Bob unfortunately have some adversaries that are working against them—namely Eve and Mallory.

Another person that we will occasionally use in our examples is a gentleman by the name of Trent. Trent is a trusted third party. In particular, Trent is trusted by Alice and Bob. Alice and Bob can rely on Trent to help them get some of their work accomplished. We will provide more details about Alice, Bob, Eve, Mallory, and Trent as necessary, and we encourage you to learn more about them by reading "The Story of Alice and Bob" (Gordon 1984).

1.2. Authentication

Authentication is the act of verifying someone's identity. When exploring authentication with our fictitious characters Alice and Bob, the question we want to ask is: if Bob wants to communicate with Alice, how can he be sure that he is communicating with Alice and not someone trying to impersonate her? Bob may be able to authenticate and verify Alice's identity based on one or more of three types of methods: something you know, something you have, and something you are.

1.2.1. Something You Know

The first general method Bob can use to authenticate Alice is to ask her for some secret only she should know, such as a password. If Alice produces the right password, then Bob can