# Preface

This book is intended to teach general principles of computer security from an applied viewpoint. Readers of this book will learn about common cyberattacks, including viruses, worms, Trojan horses, password crackers, keystroke loggers, denial of service, spoofing, and phishing. They will learn about techniques for identifying and patching vulnerabilities in machines and networks as well methods for detecting and repairing infected systems. They will study fundamental building blocks of secure systems such as encryption, fingerprints, digital signatures and basic cryptographic protocols. Finally, they will also be exposed to the human and social aspects of computer security, including usability, interfaces, copyright, digital rights management, social engineering, and ethical issues.

This book is different from existing cryptography books, therefore, which focus on the mathematical and computational foundations of security, since this book focuses primarily on the systems, technology, management, and policy side of security. Likewise, this book is different from existing computer security books that focus on graduate-level formalisms and assume extensive knowledge of computer science. Instead, this book assumes only the most basic of prerequisite knowledge in computing, making it suitable for beginning computer science majors, as well as computer science minors and non-majors. Thus, this book can serve as a textbook for computer-security courses that are taught at the undergraduate level to help give a new generation of computer professionals an increasing awareness and knowledge of computer security.

## Approach

Teaching a course on computer security has often proven to be both controversial and challenging. A first issue is the prerequisites for such a course. Traditionally, computer security courses taught today assume extensive computer science and mathematics background, and they require as prerequisites a variety of junior/senior courses such as algorithms, operating systems, computer networks, number theory, and software engineering. The typical assumption is that, in order to start learning about computer security, students need an advanced knowledge of how computer systems function and significant abilities in programming and mathematics. This approach gives flexibility to instructors in selecting advanced topics and projects. However, it has led to small enrollments in computer security courses and to a consequent shortage of computer-security experts, since

most computer science concentrators graduate today without ever learning about computer security. Moreover, this traditional approach puts computer security courses out of reach to computer science minors and nonmajors.

Instead, we have tried to make this book suitable for a computer security course that has as its sole prerequisites an introductory computer science sequence (e.g., CS 1/CS 2). To deal with needed background knowledge, We cover computer security while at the same time providing necessary on the foundations of computing needed to understand the specific topics in computer security that we present. Thus, a course that utilizes our book as its textbook can serve the dual purpose of teaching computer-security topics such as access control, firewalls, and viruses as well as introducing a variety of fundamental computer-science concepts in algorithms, operating systems, networking, and programming languages. We believe it is possible to convey fundamental computer security concepts and give students a working knowledge of security threats and countermeasures by providing just-enough and just-in-time background computer science material for their understanding. Therefore, this book leverages and exercises a student's knowledge of programming and algorithms in the setting of information security, since both a solid programming discipline and efficient algorithms are essential for developing effective security solutions.

Another reason to make this book widely accessible is to encourage students to think about security issues and to deploy security mechanisms early in designing software applications or in making software purchase/deployment decisions. This skill will be certainly appreciated by future employers, who include leading corporations in the financial, healthcare and technology sectors, for whom the security of software applications is often a critical requirement. Besides training information technology professionals in security, this book aims to create security-savvy computer users who will have a clear understanding of the security ramifications of using computers and the Internet in their daily life (e.g., for online banking and shopping). Last, but hardly least, motivated by the recent debate on electronic voting, we desire that students become aware of the potential threats to individual privacy, and possibly to democracy itself, that may arise from inappropriate use of computer security technology.

## Computer Science Concepts

General topics for the book and related general computer science concepts are shown in Table 1.

| Topic | Subtopic | Related Concepts |
|---|---|---|
| Code execution vulnerabilities | buffer overflow, sandboxing, mobile code | programming languages, software engineering |
| Virus protection | trojans, viruses, and worms | pattern matching |
| Access control | users, roles, policies | operating systems |
| Authentication and encryption | cryptosystems, hashing, signatures, certificates | computational complexity |
| Network security | SYN flooding, connection spoofing, firewalls, denial of service, TLS, intrusion detection, virtual private networks | network protocols |
| Human and social issues | usability, social engineering, copyright and digital rights management | user interfaces, ethical issues |

**Table 1:** Book topics and related general computer science concepts.

## Chapter Listing

A listing of the chapters of this book is given in Table 2.

1. Introduction
2. Physical Security
3. Operating Systems Security
4. Malware
5. Network Security
6. Browser Vulnerabilities
7. Cryptography
8. Security Models
9. Security Practice
10. Applications Security

**Table 2:** List of chapters.

## Supplementary Materials

We have created a set of teaching materials, which will help instructors to teach the course. In particular, we have developed the following:

- A collection of slide presentations (in a standard file format compatible with both Microsoft PowerPoint and OpenOffice Impress), each suitable for a one-hour lecture, covering all the course topics. The pre-

sentations will include links to relevant resources on the web and will have extensive notes.

- Fully developed programming projects, including projects on the following topics:
  1. virus and worm propagation,
  2. firewalls,
  3. cryptography and digital rights management,
  4. web applications.

  Each project stimulates the student's creativity by challenging them to either break security or protect a system against attacks.

## Projects

In terms of developed projects, we have a collection of projects to be used both in courses focused on computer security and in courses that cover topics related to computer security. A wide set of options will allow instructors to customize the projects to suit a variety of learning modes and lab resources.

**Approach.** The detection of vulnerabilities in computer systems and the development of defense mechanisms to remove such vulnerabilities are two fundamental activities for computer security professionals. Our proposed projects explore the interplay between these activities by offering students creative, hands-on experience in them. In each of our proposed projects, the student is given a realistic, though simplified, version of a working system with multiple vulnerabilities and a list of allowed attack vectors. The student has to analyze the system to identify and remove its vulnerabilities.

Each projects is designed to allow for multiple ***phases*** of work. In turn, each phase of the project has one of the following two ***modes***:

- In ***break-it*** mode, the student attacks the system by developing exploits that take advantage of the discovered vulnerabilities of the system.
- In ***fix-it*** mode, the student hardens the system by developing mechanisms (e.g., software patches) for removing or mitigating the discovered vulnerabilities.

We offer three project options, depending on the number of phases. A ***simple*** project has only one phase, which can be either in break-it mode or fix-it mode. An ***intermediate*** project has two phases. For example, an intermediate "break it then fix it" project is structured as follows: in the first phase, the student operates in break-it mode and submits one or more exploits; in the second phase the student operates in fix-it mode and submits

patches that prevent the previously found exploits. An ***advanced*** project has three phases. For example, an advanced "break it then fix it then break it" project starts as the intermediate "break it then fix it" project above. Next the students work in break-it mode attacking the hardened systems submitted by other students.

While it is natural to start an intermediate or advanced project with a break-it phase, a project can also start with a fix-it phase. In particular, discuss below how specific projects can be structured as "***fix it then break it***".

The projects available at the companion website are outlined below. For each project, we discuss the its goals, coverage of concepts, support software and project configuration options.

**Sentinel.** The project code named Sentinel covers the subject of ***malware*** (malicious software), such as viruses and worms. In particular, the project addresses infection methods, propagation methods, concealment methods, detection and removal of malware. The support software we plan to provide consists of a fully functional, though benign, educational worm plus tools for detecting and removing the worm. Sample project configurations are outlined below.

- ***Fix it*** Sentinel. This configuration of the project consists of a single fix-it phase. The student is given a virtual machine compromised by the worm. The student conducts a forensic analysis to identify the worm's infection and propagation methods and payload and develops an automated tool for detecting and removing the worm. In this project configuration, the student has access only to the executable code of the worm. This project configuration can be made harder by enabling a concealment method for the worm.
- ***Break it*** Sentinel. This configuration of the project consists of a single break-it phase. The student is given portions of the worm's source code and a set of virtual machines to infect. The student develops the missing components of the worm (e.g., the propagation and concealment methods). This project configuration can be made harder by making the student implement larger, or more complex, portions of the worm's code.

While focusing on malware, the project also addresses fundamental topics in programming languages (e.g., program execution, exploited in buffer overflow attacks), operating systems (e.g., processes, services and boot sequence, used for execution and persistence of the worm), cryptography (e.g., encryption, used for concealment), algorithms (e.g., pattern matching, to detect virus signatures, and graph traversals, to model propagation).

**Spidernet.** The project code named Spidernetcovers denial-of-service attacks and firewalls. In particular, this project addresses basic denial-of-service attack methods such as flooding (e.g., SYN flood, ping flood and smurf attacks) and malformed packets, and the architecture and operation of a firewall. The support software we plan to provide for this project consists of a ***packet monitor plugin*** program that controls and extends the functionality of the standard Linux firewall and implementations of popular denial-of-service attacks. The purpose of the packet monitor plugin is to hide the details of intercepting and parsing packets, and to allow students to focus on the essential filtering operation of the firewall. Sample project configurations are outlined below.

- ***Fix it*** Spidernet. This configuration of the project consists of a single fix-it phase. The student is given $(i)$ a victim virtual machine, running a web server; $(ii)$ one or more attacker virtual machines, each equipped with software to launch a denial-of-service attack on the victim; and $(iii)$ portions of the packet monitor plugin code. The student experiments first with the denial-of-service attack to identify its method, using a packet sniffer. Next, the student adds filtering code to the packet monitor plugin to mitigate the attack. For example, to counter a SYN-flood attack, the student can implement a simple per-host connection limit, or a more sophisticated cryptographic method, such as SYN-cookie.
- ***Fix it then break it then fix it*** Spidernet. This configuration of the project starts with a fix-it phase where the student is given $(i)$ the victim virtual machine and $(ii)$ portions of the packet monitor plugin code. However, the student does not have access to the attacker virtual machines in this phase. The student hardens the victim machine by adding filtering code aimed at defending against various types of possible denial-of-service attacks. Next, in the break-it phase, each student is given $(i)$ one or more attacker virtual machines; $(ii)$ portions of the denial-of-service code; and $(iii)$ copies of the hardened machines developed by the other students. The student completes the denial-of-service code and launches attacks on the other student machines. Based on the results of the attacks in the second phase, each student gets a second chance to further harden their machine in the final fix-it phase.

While focusing on denial-of-service attacks and firewalls, the project also addresses fundamental topics in computer networks (e.g., the IP, ICMP and TCP network protocols exploited in the attacks), basic cryptography (e.g., one-way functions and their efficient design, used in the SYN-cookie mitigation method), and algorithms (e.g., fast data structures for dictionaries, such as hash tables, used in the rate limitation approach).

**Player.** The project code name Playercovers ***digital rights management*** (***DRM***) in the context of online music distribution. The support code for this project implements an online music distribution system consisting of an online music store and a DRM-equipped music player. The system models, in a much reduced scope, the basic operations of commercial online music systems, such as Apple's iTunes. The player allows users to login to the store, download songs from the store, and play them. The player stores the songs on the user's machine in encrypted form and decrypts them when playing.

The support code is customizable to include multiple vulnerabilities in the system, illustrating fundamental DRM techniques. Examples of such vulnerabilities include passwords or songs transmitted unencrypted over the network, songs temporarily stored unencrypted on the user's machine, flawed authentication protocols for user authentication and for music download by the player, short keys used to encrypt songs, weak passwords to access the store, and player's executable code that is easy to reverse engineer and modify.

A sample project configuration is outlined below.

- ***Break it then fix it*** Player. In the first phase, operating in break-it mode, the student is given the executable code of the player and access to a server running the music store application, thus allowing the student to interact with the system. However, in this phase, the student does not have access to the source code of the system. The students identifies vulnerabilities in the DRM features of the system and implements exploits demonstrating them. For example, to exploit the short-key vulnerability, the student can write a brute-force decryption attack over the entire key space, while to exploit the unencrypted temporary song file vulnerability, the student can change the permissions of the directory of temporary files used by the player to prevent the removal of the unencrypted song file. In the second phase, the student is given the source code of the system and access to the exploits implemented by the other students. The student strengthens the system by correcting the vulnerabilities as much as possible.

While focusing on digital rights management systems, this project also addresses fundamental topics in operating systems (such as passwords and file permissions), intermediate cryptography (such as symmetric encryption, public-key encryption, collision-resistant hash functions and key distribution protocols), and computer networks (such as eavesdropping network traffic).

**Wargames.** The project with code name Wargamescovers the subject of web application security in the context of online multiplayer games. The

security concepts illustrated in this project are also applicable to auction websites and stock trading websites. The support code for the project is an implementation of a multiplayer interactive online game allowing players to compete in a virtual automobile race. The system consists of a game server, a client application, and a scoreboard server. Players create a new game or join an existing game by connecting to the game server. The game server maintains the state of a game by providing a shared virtual world to the client applications of the players and generating random events affecting the game. Periodically, the game server reports data on the game to the scoreboard server, which maintains and displays statistics about the games and the players. While the goal of the system is to ensure fairness in the individual games and accurate statistical reporting, the support code is customizable to include multiple vulnerabilities in the system, such as session hijacking, escalation of privilege, injection attacks (e.g., SQL code injection), replay attacks, man-in-the middle attacks, and collusion attacks. For example, a predictable pseudo-random number generator in the game server may be exploited by a player to anticipate events and gain unfair advantage, or an SQL injection vulnerability in the scoreboard server may allow a player to unfairly reach top rank.

A sample project configuration is outlined below.

- ***Break it then fix it*** Wargames. In the first break-it phase of the project, the student is given the source code of the system. The student identifies and analyzes the security related portions of the code with the goal of discovering vulnerabilities and develops exploits to illustrate them. In the second phase, the student is given access to the exploits discovered by the other students. The student then develops revisions to the code to eliminate or mitigate the discovered exploits.

While focusing on web application security, this project also addresses fundamental topics in programming languages (e.g., program verification, to ensure the correct processing of game events), cryptography (e.g., pseudo-random number generators), and network protocols (e.g., HTTP and SSL).

**Topic Coverage Grid.** We show in Table 3 below the broad security topics emphasized by each of the proposed projects and the use of virtualization.

## Prerequisites

We have written this book assuming that the reader comes to it with certain knowledge. That is, we assume that the reader is at least vaguely familiar with a high-level programming language, such as C, C++, or Java, and that he or she understands the main constructs from such a high-level language.

| **Project** | Operating Systems | Networks | Programs | Cryptography | Virtualization |
|---|---|---|---|---|---|
| Sentinel | • | | • | | required |
| Spidernet | | • | | • | required |
| Player | • | • | • | • | not needed |
| Wargames | | • | • | • | not needed |

**Table 3:** Broad security topics addressed by the projects and their virtualization requirements.

In addition, we assume the reader has a familiarity with the fundamental concepts of basic data structures and computer systems.

## About the Authors

Professors Goodrich and Tamassia are well-recognized researchers in computer security, algorithms and data structures, having published many papers in this field, with applications to computer security, cryptography, Internet computing, information visualization, and geometric computing. They have served as principal investigators in several joint projects sponsored by the National Science Foundation, the Army Research Office, and the Defense Advanced Research Projects Agency. They are also active in educational technology research.

Michael Goodrich received his Ph.D. in Computer Science from Purdue University in 1987. He is currently a Chancellor's Professor in the Department of Computer Science at University of California, Irvine. Previously, he was a professor at Johns Hopkins University. He is an editor for the *Journal of Computer and Systems Sciences* and and *Journal of Graph Algorithms and Applications*. He is a Fulbright Scholar, an ACM Distinguished Scientist, and a Fellow of the American Association for the Advancement of Science (AAAS) and the Institute of Electrical and Electronics Engineers (IEEE).

Roberto Tamassia received his Ph.D. in Electrical and Computer Engineering from the University of Illinois at Urbana-Champaign in 1988. He is currently professor and chair of the Department of Computer Science at Brown University. He is editor-in-chief for the *Journal of Graph Algorithms and Applications*. He previously served on the editorial board of *Computational Geometry: Theory and Applications* and *IEEE Transactions on Computers*. He is a Fellow of the Institute of Electrical and Electronics Engineers (IEEE).

In addition to their research accomplishments, the authors also have extensive experience in the classroom. For example, Dr. Goodrich has taught data structures and algorithms courses, including Data Structures as a freshman-sophomore level course, Applied Cryptography as a sophomore-

junior level course, and Internet Algorithmics as an upper level course. He has earned several teaching awards in this capacity. His teaching style is to involve the students in lively interactive classroom sessions that bring out the intuition and insights behind data structuring and algorithmic techniques. Dr. Tamassia has taught Data Structures and Algorithms as an introductory freshman-level course since 1988 and has recently introduced a new freshman-level computer security course. One thing that has set his teaching style apart is his effective use of interactive hypermedia presentations integrated with the Web.

The instructional Web sites, datastructures.net and algorithmdesign.net, supported by Drs. Goodrich and Tamassia, are used as reference material by students, teachers, and professionals worldwide.

## Acknowledgments

There are a number of individuals who have made contributions to this book.

We are grateful to all our research collaborators and teaching assistants, who provided feedback on early drafts of chapters and have helped us in developing exercises and projects. In particular, we would like to thank Matt Goldstein, and others.

We are also truly indebted to the outside reviewers and readers for their copious comments, emails, and constructive criticism, which were extremely useful.

The team at Addison-Wesley has been great. Many thanks go to the team members.

The computing systems and excellent technical support staff in the departments of computer science at Brown University and University of California, Irvine gave us reliable working environments. This manuscript was prepared primarily with the LaTeX typesetting package.

Finally, we would like to warmly thank Isabel Cruz, Karen Goodrich, and our parents for providing advice, encouragement, and support at various stages of the preparation of this book. We also thank them for reminding us that there are things in life beyond writing books.

Michael T. Goodrich
Roberto Tamassia