Chapter 1
Introduction

Revision 1.0, March 2026.

Creating a good, clean, efficient software system is difficult. It is more than just clever coding of algorithms; it requires careful design and engineering for the system to be secure. A lot of thought and work goes into creating a software system that is robust and reliable. Creating software that is also resistant to attack is much more difficult. Such resistance to attack requires awareness of the potential threats, knowledge of good coding practices, knowing how to use the right tools, and careful attention at each stage of the software lifecycle.

The difficulty in creating secure software is underscored by the steady stream of headlines reporting on serious security breaches in software systems big and small. Even the biggest software companies – the ones with well trained and dedicated security staffs – are attacked frequently and successfully. Our goal with this book is to keep you and your organization off the front page of your local newspaper, or even the national and world newspapers.

1.1 The Path to this Book

We did not set out to write a book on software security. Our path to this book started many years ago when we started helping software teams assess the security of their software. Over the course of performing in depth vulnerability assessment of a variety of software projects, we started to notice a common set of programming and design mistakes that made the software vulnerable to exploit. As we started to build this collection of commonly-found mistakes, we also started to prepare tutorial material on how to avoid each of them.

We used this tutorial material at conferences and workshops, corporate sites, and government labs to teach software practitioners how to identify these design and coding mistakes, and show the software practitioners ways to avoid them. Our collection of commonly-found mistakes grew as we assessed more software projects, so our tutorial materials grew alongside it. We expanded our coverage of topics beyond basic secure coding to include the major steps in the secure software design lifecycle.

Beyond secure coding, we included secure software design principles, threat modeling, software assessment tools, testing tools and in-depth vulnerability assessment techniques.

1.2 How to Use this Book

There are several ways that this book can be useful. Below, we outline a few of these ways. As an instructor, this section can be helpful in deciding how to incorporate these materials into your course and how to design a course specifically on software security.

As the foundation for a software security course:

Taken as a whole, this material forms curriculum for an advanced undergraduate or beginning graduate course. Our course, CS542, Introduction to Software Security, at the University of Wisconsin-Madison is one such course.

The book progresses along the steps in the software development lifecycle. After a couple important background chapters (Chapters 2 and 3), we moved into the design stage of the software (Chapters 4 through 8). Students in a university computer science program get little if any exposure to the design of software. Most programming assignments in a computer science class are based on implementing something that the instructor specified. If the students are building a larger piece of software, such as a compiler, the sequence of assignments directs the student through each stage of development without significant design effort on the part of the student.

The next section on secure coding (Chapters 9 through 25) is often the most popular with the students, as it pertains to the familiar activity of coding. This section covers a wide variety of topics to give the students a good foundation for design and implementation scenarios that can lead to security vulnerabilities and the strategies to avoid or fix them. Note that our list of scenarios originates from our vulnerability assessment activities, reflecting the kind of mistakes that we observed in real, deployed code. This list nicely correlates with other lists of common mistakes, such as the MITRE list of the Top 25 CWEs (coding flaws) found in reported vulnerabilities.

We then move to a section covering defensive techniques that support secure code. Chapters 26-28 cover ASLR, memory safety checks, and control flow integrity checks, reflecting important developments by the architecture, operating system, and compiler communities to reduce the likelihood of a successful exploit. These topics are important for any software professional to understand what protections are being provided to them and what are the limitations of these protections.

Just as secure design should come before coding, vulnerability assessment is the evaluation process after the software is running to find and eliminate any design or coding vulnerabilities that are in the code. It has many similarities to the threat modeling process, with the reference point being the existing code instead of a proposed design. It is especially interesting for students to

compare these two efforts side-by-side to provide a better understanding of each one.

The book finishes with a section on tools that a programmer or analyst can use to help make their code more secure. Chapters 36 and 37 focus on static tools and Chapters 38-43 focus on dynamic tools. Each of these chapters covers a topic that will make a programmer more productive in developing secure code.

As support for other computer science curriculum:

Security should not be a separate afterthought, only included in a specific course on the topic. The ideas should permeate everything we learn about software design and coding.

As such, the chapters in this book can be used selectively to support a variety of classes. For example, for a student taking an operating systems course, where the programming assignments are often written in C, Chapter 9 on pointers and strings can be quite useful. For a student in a database class, Chapters 15 and 16 – which introduce injection attacks and describe SQL injections – can serve as a good reference for the topic. For a class in numerical analysis, Chapters 10 and 11 on numeric errors can bring a different viewpoint on the topic. For a class in web programming, the whole sequence in Chapters 20-25 can provide the class with the security view of what they are learning. Even in an IT management course, Chapter 35 on how to handle software vulnerabilities can be directly applicable to that topic.

The basic idea is that you cannot mention security often enough so look to incorporate in incrementally across a whole curriculum.

As a reference and learning aid for the professional programmer:

If you are practicing professional, whether you are designer, developer, or security analyst, you are confronted with an incredibly wide variety of topics to master. Your education and work may have given you experience with only a limited number of the topics confronting you. This book is intentionally divided into chapters or sections that are relatively independent. As such, you can pick and choose among them, as you need to reinforce your background in a certain area.

If you have the time, it is worth reading the early chapters (Chapter 2-4) to provide a foundation and structure for the other things that you will learn.

1.3 Let's Get Started

There is no time like the present to start thinking about security. So pick a chapter and get started. And please send us feedback on your experiences using this book. We listen and try to improve with each thing that we hear.