

---

## Preface

---

A few years ago, before J2EE (**Java 2 Platform, Enterprise Edition**) became such a dominant platform for building enterprise systems and long before Web services became central to the IT<sup>1</sup> strategy of every small and big company, I was tasked with helping a small company use one of our products more effectively. This company, which must remain unnamed for reasons of privacy and professional conduct, was setting up an infrastructure for creation of a dynamic and collaborative community of businesses so that their people and systems could exchange digital content and information over the Internet in the most appropriate, secure and timely manner. Our sales and marketing department did a good job in convincing them that our soon to be released product, let us call it *ProdX*, was built to satisfy exactly the same requirements. After numerous technical meetings and the promise of premium customer status, free technical support, training and unrestricted access to the development team, they agreed to use ProdX.

ProdX was built and promoted as a Java-based middleware product suite with a strong and unique security architecture for allowing companies to do business over the Internet. However, few people outside the security development team, a sub-team of the overall ProdX development group, understood this architecture well and even fewer knew how to use its APIs effectively or how to set it up for *data center* operations. Developers, managers and operations staff of the customer company had numerous meetings, conference calls and e-mail exchanges, either through me or directly, with the security development team. And still, they did not feel very comfortable.

At that time, security wasn't the focus of my primary job and I must confess that I was also having difficulty in comprehending certain aspects of ProdX in the context of its use. Watching these interactions, it became very obvious to me that the security team had a sound cryptographic background and were deeply involved in developing state of the art security theory and standards, but had little appreciation of the fact that our customers were more interested in having their developers know what APIs to use, how, when and where to use them and having their operations people know how to work out step by step processes and procedures for routine and emergency operations.

Eventually, they did get what they wanted and were able to go live with ProdX. However, we all felt that the whole thing took a lot more time and attention than required.

Since then I have spent a lot more time working with J2EE-based products and Web services infrastructure software. As an architect, I have also participated in the development

---

<sup>1</sup> IT, or Information Technology, is usually referred to the technologies for processing, storing and transporting information in digital form.

of Java standards for Web services, reviewed many software products in these areas and interacted with many customer organizations and listened to their security, performance and other concerns. In the meanwhile, the Java platform, its security architecture and APIs have continuously evolved and matured. However, none of this has eliminated the gap between what is available and what is in use.

I attribute this to many factors. The reality is that some of the technology is very new, and at times, quite complex. At the same time, the changing ways of using the Internet for business-critical operations and the increased threat of a security breach have kept practitioners at their toes. This constant churn at both ends has kept the gap alive and kicking.

It is the aim of this book to narrow this gap, at least in the area of J2EE-based Web applications.

## **J2SE, J2EE AND APPLICATION SECURITY**

The life of a Java professional had never been more *fun*. Besides the traditional forms of enterprise application and Web application development, the emergence of XML and Web services technologies has resulted in a new Web-based distributed computing paradigm, with its own set of design, development, deployment and operations challenges. This is matched, in almost equal measures, by the growing richness of the Java platform, consisting of both the Standard Edition (J2SE) and the Enterprise Edition (J2EE), making it an apt toolchest for an increasingly complex world.

This toolchest has drawers filled with APIs, patterns, tools and conventions for different environments and different needs, waiting to be used at *the right place*, at *the right time*, and in *the right way*. Multiple implementations of the same APIs, sometimes from different vendors but more often freely available from the Open Source Community, allows one to pick the best of breed for a particular purpose. It is this multitude of choice and freedom that makes the life of a Java professional *fun*.

It is often claimed that Java is designed for secure programming from the ground up and security features are not added as an after thought. And indeed, it is quite unique in its ability to declaratively specify what a piece of code can and cannot do. Support for cryptographic operations and public key infrastructure through Java Cryptographic Architecture in J2SE is also quite remarkable. In addition, J2EE defines security characteristics for distributed processing, data access, transactions, management and other such aspects.

All this makes Java an excellent platform for constructing secure enterprise applications.

## **SCOPE OF THE BOOK**

This book is about applying security concepts, techniques, APIs, standards and tools to identify and address enterprise application security problems within the Java environment.

You will find the contents of the book useful for all stages of development lifecycle -- analysis, design, development, deployment and operations.

Personally I have enjoyed reading books that provide insight into the subject matter with appropriate focus on *whys* and *hows*, turning to official standards or product manuals for detailed and very specific information. I also like to see source code fragments, execution steps and screen shots wherever appropriate, for they tell me exactly what to do to accomplish a desired result. Needless to say, this book has been written with these principles in mind.

The main focus of this book is the security of data and information maintained and served by enterprise applications running under J2EE. We accomplish this by identifying what needs to be secured, how and where. Further, we discuss the different mechanisms to accomplish this, covering:

- **Cryptographic concepts and services** that are at the heart of many security APIs and features.
- **Public Key Infrastructure** that makes cryptography as basis of trust for security applications.
- **Access Control** based on the origin of code, signer of the signed code, and/or the credentials of the user running the code.
- **Secure communication of data** using Secure Socket Layer, also known as Transport Layer Security.
- **Integrity, Authentication and Confidentiality of XML messages** using XML Signature and Encryption.
- Security characteristics of **RMI-based distributed applications**.
- Securing **Servlet and JSP-based Web Applications**.
- Security of **EJB-based Enterprise Applications**.
- **Security aspects of Web services** development, deployment and operation.

Enterprise application security in J2EE builds upon the foundation of security concepts and architectures such as Cryptography, Digital Certificates, Public Key Infrastructure, Java security model, Java Cryptographic Architecture etc. One should be comfortable with these topics to follow the main text. Similarly, one should know about basic Web services interoperability standards such as SOAP and WSDL and the Java programming model for Web services.

Not assuming that every reader is current with all these technologies, we cover them briefly, stressing those aspects that are more pertinent for the main subject area. This coverage is more appropriate as a quick refresher than a basic introduction and should be used accordingly.

At the same time, we must acknowledge that computer and network security is a vast and expanding field incorporating such diverse topics as cryptography, operating system security, network security, firewalls, computer viruses and anti-virus software, intrusion

detection, incident response, vulnerability analysis, biometrics, social engineering, privacy and legal aspects, trusted computing, and so on. Though we recognize the importance of these topics in comprehensive security planning, they are not the focus of this book and hence find only brief overview in the first chapter.

We also refrain from getting into details of product specific non-standard security features. The only exceptions are product features that help illustrate a specific point not covered by the standards.

## WHO SHOULD READ THIS BOOK

This book is primarily written for

- **Java programmers** developing Java applications.
- **System administrators** managing J2EE-based applications.
- **Architects** evaluating security products from different vendors and architecting secure Java solutions.
- **Project Managers** planning, managing and overseeing Java and J2EE projects.

Specifically, this book is not targeted at security experts designing security protocols, APIs and products. Intruders looking at devising ways to compromise security will also be disappointed.

## ORGANIZATION OF THE BOOK

This book is organized in three main parts. Part one is more like a refresher on basic security and the Java platform. If you are already familiar with these topics, feel free to move over to part two. You could also choose to read certain sections selectively, and in any order.

Part two introduces the basic building blocks of the Java platform's security architecture – APIs for cryptographic operations, Public Key Infrastructure, access control mechanisms, Java Secure Socket Extension for secure communication, and APIs for XML Signature and XML Encryption. A sound understanding of these topics is a must for developing secure enterprise applications.

Part three ties the concepts introduced in part two to specific J2EE APIs – RMI, Servlets, EJBs and Web services – and their security architecture. The emphasis is on getting hands-on exposure to APIs and products, aided by lots of working code.

Part two and three are the main reason this book exists. Here we cover the underlying technology, identify the security issues in typical J2EE applications and go on to address

them, explaining the abstractions, standards, protocols and APIs. An important aspect of this part is that real, best-of-breed products are used to illustrate the concepts.

Below is an outline of the book's parts and chapters.

## Part One: The Background

Part one builds the necessary background on computer security and the Java platform, preparing the reader for the discussion in the later chapters.

**Chapter 1. *A Security Primer*:** This chapter looks at news reports, survey findings and case studies to get a feel for computer and network security problems. This is followed by a review of the technologies behind the Internet and the corporate IT infrastructure and a discussion of how attackers exploit vulnerabilities to mount attacks. The chapter is concluded with brief descriptions of enabling technologies in the fight against computer crime and how application security, the main topic of the book, fits into the overall scheme of things.

**Chapter 2. *A Quick Tour of the Java Platform*:** This chapter is a backgrounder on the Java platform, consisting of J2SE and J2EE, with a focus on security aspects. As with the previous chapter, the emphasis is on understanding the broader context of various Java technologies and their relationship to security-specific technologies.

## Part Two: The Technology

Part two of the book focuses on the basic security technology available within the Java platform. Most of these technologies and APIs are packaged with J2SE and form the basis for the security capabilities of various enterprise application development APIs such as RMI, Servlets, EJBs and Web services.

**Chapter 3. *Cryptography with Java*:** This chapter explains cryptographic services and the Java API supporting these services. Basic cryptographic APIs JCA (**Java Cryptography Architecture**) and JCE (**Java Cryptography Extension**) are covered. In this chapter, you will learn about the secret key and public key cryptography, message digests, Message Authentication Code, and digital signature. The performance of certain cryptographic operations is also analyzed.

**Chapter 4. *PKI with Java*:** This chapter discusses Java support for PKI (**Public Key Infrastructure**) components such as X.509 certificates, CAs (**Certification Authorities**), CRLs (**Certificate Revocation Lists**), and repositories in the Java platform. Steps in obtaining CA signed certificates and managing certificates in a keystore are explained. It also covers the `keytool` utility for manag-

ing private keys and certificates. Java APIs to handle digital certificates and certification paths are also covered.

**Chapter 5. *Access Control*:** This chapter explains the security model to protect resources within a JVM secured with a Security Manager. Code centric and user centric access control through policy files and JAAS (**Java Authentication and Authorization Service**) is also covered. A sample application with moderate complexity is introduced and JAAS is used to secure this application.

**Chapter 6. *Securing the Wire*:** This chapter explains SSL (**Secure Socket Layer**), also known as TLS (**Transport Layer Security**), protocol for securing exchange of information over unprotected networks at the transport level. Java API JSSE (**Java Secure Socket Extension**) to develop SSL enabled client and server programs are explained and illustrated through example programs.

**Chapter 7. *Securing the Message*:** This chapter talks about message security as a means to secure messages independent of transport. XML security standards XML Signature and XML Encryption are explained. Two libraries with programmatic support for these standards are covered: Verisign's TSIK (**Trust Services Integration Kit**) and Infomosaic's SecureXML.

## Part Three: The Application

Part three is about applying security APIs, concepts and tools to J2EE applications.

**Chapter 8. *RMI Security*:** Discusses the security issues in developing RMI based distributed applications. Covers the use of security manager to limit privileges of downloaded code, SSL for transport level security and JAAS for user authentication and access control. These techniques are further illustrated with help of examples.

**Chapter 9. *Web Application Security*:** This chapter talks about the different forms of declarative and programmatic security for Servlets and JSPs. Apache Tomcat is used to illustrate example programs. Detailed steps to setup Tomcat for accepting HTTPS connections with or without client authentication are presented. Common Web application vulnerabilities such as cross-site scripting, command injection, failure to validate input etc. and mechanisms to safeguard against these issues are also covered.

**Chapter 10. *EJB Security*:** This chapter discusses how EJB architecture facilitates development of software components for assembling secure enterprise applications. BEA's *WebLogic Server* is used to explore security concepts such as JNDI-based client authentication, SSL for transport-based security, security protection domain spanning multiple J2EE servers, declarative access control

through deployment descriptors, programmatic access control APIs, identity propagation, identity delegation and other related concepts.

**Chapter 11. *Web Services Security*:** This chapter talks about security issues in developing, deploying and invoking Web services with JAX-RPC APIs. Open source SOAP (**Simple Object Access Protocol**) engine Apache Axis is used to illustrate the APIs and the examples. It explains use of a number of technologies for Web services security: Servlet deployment descriptor and API for authentication and access control, SSL for transport-based security and WS Security for message-based security. JAX-RPC-compliant SOAP handlers are developed using VeriSign's TSIK and WSSecurity class library to illustrate secure Web service examples.

**Chapter 12. *Conclusions*:** This final chapter takes a step back and reviews the subject matter of the book from a distance, identifying patterns, general principles and the interconnectedness of the topics. Security issues in various Java-based enterprise application development infrastructure, such as sockets, RMI, Servlets, EJBs and Web services, are summarized and their dependence on lower level cryptographic services, PKI entities, security protocols, authentication services etc. is analyzed.

## Appendices

A number of appendices supplement the subject matter of the book.

**Appendix A. *Public Key Cryptography Standards*:** A brief overview of relevant PKCS standards.

**Appendix B. *Standard Names --- Java Cryptography Services*:** Standard names used for cryptographic service algorithms and types in Java Security APIs.

**Appendix C. *JSTK Tools*:** A brief user guide to the JSTK (**Java Security Tool Kit**) tools, the software bundled with the book.

**Appendix D. *Example Programs*:** A list of all the examples presented in the book with brief descriptions.

**Appendix E. *Products Used*:** A list of all the software products used for developing the example programs.

**Appendix F. *Standards Bodies*:** A brief description of standardization bodies related to security, Java and XML technologies.

## TYPOGRAPHIC CONVENTIONS USED IN THIS BOOK

Following typographic conventions are used in this book:

- Normal text is in Times font with 10pt size.
- The first instance of new terms, especially the ones that are explained or illustrated in the nearby text, are *italicized*.
- Certain terms are *italicized* for emphasis.
- Well-known technical acronyms are expanded in their first use, the expansion being **bold** and placed within parentheses immediately following the acronym. Example: TLS (**Transport Layer Security**).
- Java package, class and interface names occurring within the text are in New Courier font.
- Name of executable utilities, occurring within normal text, are in **New Courier bold**. Example: **keytool**.
- Commands with multiple words are also in New Courier, and have been enclosed within double quotes, as in "certtool setupca help".
- Code fragments consisting of multiple statements are in New Courier font with reduced size of 8pt.
- Line continuations, either within a command or in source files are indicated by a trailing backslash.
- Reference to a Section, Chapter, Appendix, Figure, Example or Listing within the main text is marked by *italicized title*.
- A word or term to be replaced by another word or term in actual source code, pathname or command is in *italicized New Courier*. Example: *tomcat-home\webapps*.
- Figures, large source code listings and program output blocks are labeled with brief descriptions in 8pt Helvetica.
- Interactive sessions consisting of user input and computer output are in 8pt New Courier, with the difference that all input typed-in by the user is in **8 pt. New Courier bold**.

At places, you may find other, less frequently used, conventions.

## JSTK (JAVA SECURITY TOOL KIT)

I wrote a number of programs while authoring this book. In the beginning, each program was independent, to be compiled and executed separately. However, in the course of writing new chapters and experimenting with newer concepts, I found myself repeatedly going back to these programs and tweaking them to do something useful by changing certain hard-coded values. It is then that I realized that the independent programs were good for illustrating concepts in isolation, and hence inclusion in the book, but were not easy to use as building blocks for carrying out real tasks.

As the number of independent programs grew, it became harder for me to remember what each program actually did. Modifying them for the current task at hand was also not very effective. One weekend, in frustration, I sat down to combine a set of related programs as command line utilities with a common way of handling arguments, displaying help messages and handling errors. This turned out to be a massive re-factoring exercise, with a completely revamped directory structure, Apache Ant-based build infrastructure and a nicely integrated suite of tools. I started calling the complete package **JSTK (Java Security Tool Kit)** and shared it with some of my friends and colleagues. They, too, found it useful – not because it illustrated how to use certain classes but because it did something they needed to get done.

The utility programs were pretty cool but too complex to be used in the book as programs to illustrate individual concepts or APIs. Some of the utilities had a significant amount of code that was not related to the topics covered in the book. On the other hand, not every example program could logically fit within a JSTK utility. As a result, I ended up with both the example programs and the utilities in JSTK. The example programs can be found within a separate sub-tree, rooted at `jstk-home\src\jsbook`, where `jstk-home` is the installation directory of JSTK. Source files for the utilities are within `jstk-home\src\org\jstk` sub-tree. The examples are grouped together by chapter and can be built and executed individually using scripts kept in the same directory as the source files. For example, the source files and scripts corresponding to Chapter 7, *Securing the Message*, can be found in the directory `jstk-home\src\jsbook\ch7`. An Example is labeled by specifying the chapter number and the example subdirectory. For example, *Example ch9-rmb* refers to the RMB (**Rudimentary Message Board**) example of Chapter 9. A listing of all the example programs with brief descriptions can be found in the Appendix *Example Programs*.

Within JSTK, you will also find the instructions to compile and run the examples along with the source and script files. These instructions do not always match with those in the text of the book, for the purpose of the instructions in the book is to explain the underlying concepts whereas the electronic version is written for ease-of-use in compiling and running them.

The utility source files are compiled using an Apache Ant build file `build.xml` kept in the installation directory of JSTK. Refer to the Appendix *JSTK Tools* for more information on the tools.

A version of JSTK, frozen at the time of this book going to press, can be downloaded from this book's companion website <http://www.j2ee-security.net>.

## SOFTWARE USED FOR JSTK

A Windows 2000 machine and J2SE SDK v1.4.x from Sun Microsystems, Inc. have been used as the primary environment for developing and testing JSTK. They have also been run on a Linux system with Sun's J2SE platform. Scripts to launch utilities for both Windows and Linux platforms are included. Windows scripts have .bat extension and Linux scripts have .sh extension. Though not tested, Linux scripts should work fine on most UNIX variants.

Throughout the book, I have use Windows syntax for pathnames, environment variables and scripts. Linux or UNIX equivalents are quite obvious and hence are not illustrated separately.

For developing examples, I have used the following software:

- Apache Tomcat 4.1.18 (<http://jakarta.apache.org/tomcat>) for Servlet Container.
- Apache Axis 1.1RC2 (<http://ws.apache.org/axis>) for Web services platform.
- BEA Weblogic 7.0 SP2 (<http://www.bea.com>) for EJBs.
- Apache Ant (<http://jakarta.apache.org/ant>) for the build system.
- Verisign's TSIK 1.7 (<http://www.xmltrustcenter.org>) for XML Signature and XML Encryption.
- Infomosaic's SecureXML (<http://www.infomosaic.net>) for XML Signature.

More about these software products can be found in *Appendix E: Products Used for Examples*.

For running JSTK utilities, you don't need anything other than a plain J2SE v1.4.x installation.

Pankaj Kumar