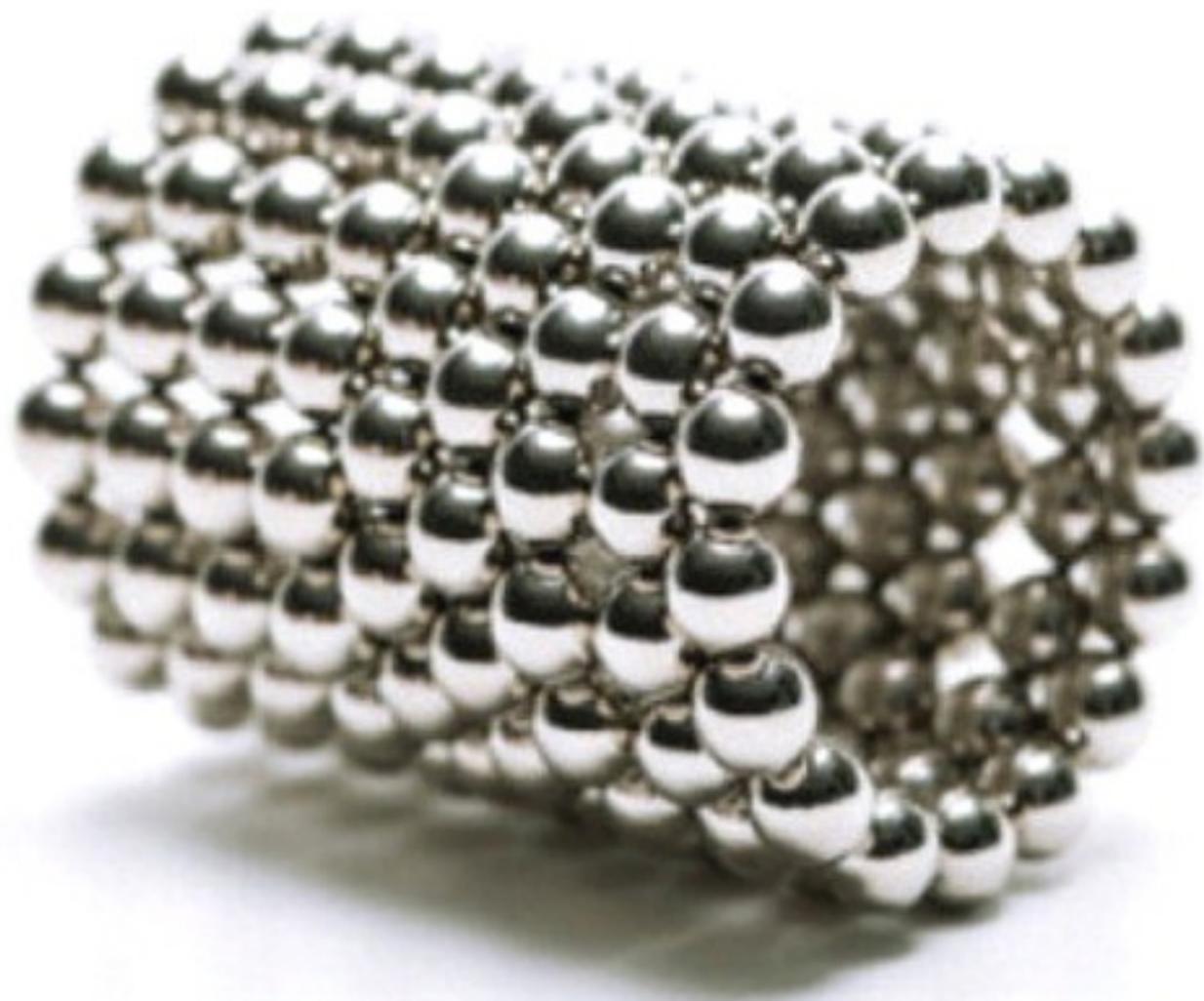


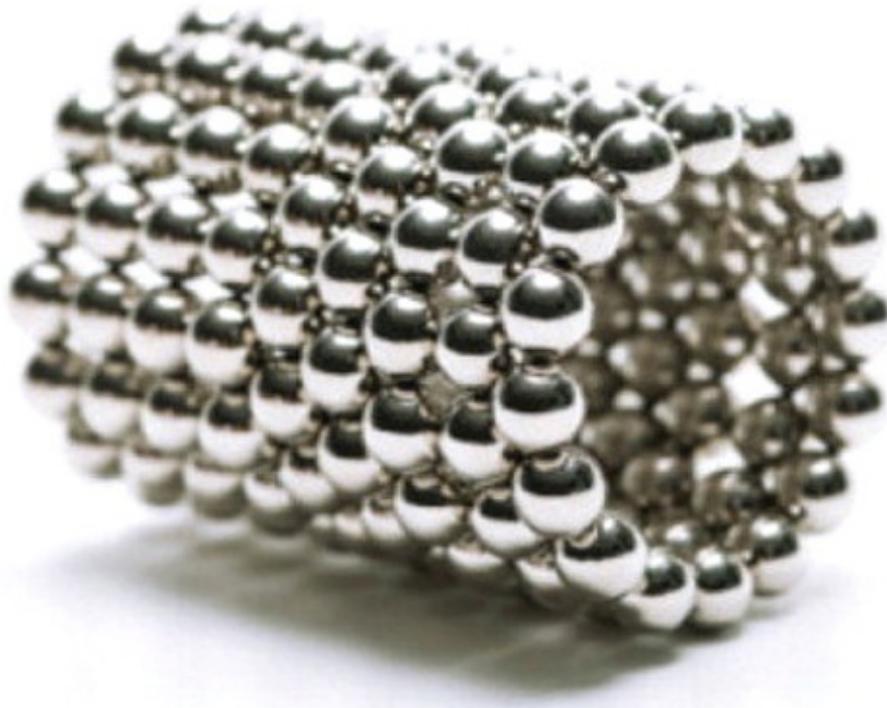
# THE CERT® ORACLE® SECURE CODING STANDARD FOR JAVA



FRED LONG | DHUV MOHLNDRA | ROBERT C. SEACORD  
DEAN F. SUTHERLAND | DAVID SVOBODA

SEI SERIES • A CERT® BOOK

# THE CERT® ORACLE® SECURE CODING STANDARD FOR JAVA



FRED LONG | DHUV MOHLNDRA | ROBERT C. SEACORD  
DEAN F. SUTHERLAND | DAVID SVOBODA

# **The CERT<sup>®</sup> Oracle<sup>®</sup> Secure Coding Standard for Java<sup>™</sup>**

**Fred Long  
Dhruv Mohindra  
Robert C. Seacord  
Dean F. Sutherland  
David Svoboda**

**◆ Addison-Wesley**

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco  
New York • Toronto • Montreal • London • Munich • Paris • Madrid  
Capetown • Sydney • Tokyo • Singapore • Mexico City

The SEI Series in Software Engineering

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

CMM, CMMI, Capability Maturity Model, Capability Maturity Modeling, Carnegie Mellon, CERT, and CERT Coordination Center are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

ATAM; Architecture Tradeoff Analysis Method; CMM Integration; COTS Usage-Risk Evaluation; CURE; EPIC; Evolutionary Process for Integrating COTS Based Systems; Framework for Software Product Line Practice; IDEAL; Interim Profile; OAR; OCTAVE; Operationally Critical Threat, Asset, and Vulnerability Evaluation; Options Analysis for Reengineering; Personal Software Process; PLTP; Product Line Technical Probe; PSP; SCAMPI; SCAMPI Lead Appraiser; SCAMPI Lead Assessor; SCE; SEI; SEPG; Team Software Process; and TSP are service marks of Carnegie Mellon University.

Special permission to reproduce portions of The CERT Oracle Secure Coding Standard for Java, © 2007–2011 by Carnegie Mellon University, in this book is granted by the Software Engineering Institute.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales  
(800) 382-3419  
[corpsales@pearsontechgroup.com](mailto:corpsales@pearsontechgroup.com)

For sales outside the United States please contact:

International Sales  
[international@pearson.com](mailto:international@pearson.com)

Visit us on the Web: [informit.com/aw](http://informit.com/aw)

*Library of Congress Cataloging-in-Publication Data*

The CERT Oracle secure coding standard for Java / Fred Long ... [et al.].

p. cm.—(The SEI series in software engineering)

Includes bibliographical references and index.

ISBN-13: 978-0-321-80395-5 (pbk. : alk. paper)

ISBN-10: 0-321-80395-7 (pbk. : alk. paper)

1. Java (Computer program language) 2. Computer security. 3. Oracle (Computer file) 4. Computer programming—Standards. I. Long, F. W. (Frederick W.), 1947- II. Carnegie-Mellon University. CERT Coordination Center.

QA76.73.J38C44 2012

005.8—dc23

2011027284

Copyright © 2012 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

ISBN-13: 978-0-321-80395-5

ISBN-10: 0-321-80395-7

Text printed in the United States on recycled paper at Edwards Brothers in Ann Arbor, Michigan.

First printing, September 2011

*To my late wife, Ann, for all her love, help, and support over the years.*

*—Fred Long*

*To my parents Deepak and Eta Mohindra, my grandmother Shashi Mohindra, and our very peppy, spotted Dalmatian Google.*

*—Dhruv Mohindra*

*To my wife, Alfie, for making this book worthwhile, and to my parents, Bill and Lois, for making it possible.*

*—David Svoboda*

*To my wife, Rhonda, and our children, Chelsea and Jordan.*

*—Robert C. Seacord*

*For Libby, who makes everything worthwhile.*

*—Dean Sutherland*

# Contents

[Foreword](#)

[Preface](#)

[Acknowledgments](#)

[About the Authors](#)

## [Chapter 1 Introduction](#)

[Misplaced Trust](#)

[Injection Attacks](#)

[Leaking Sensitive Data](#)

[Leaking Capabilities](#)

[Denial of Service](#)

[Serialization](#)

[Concurrency, Visibility, and Memory](#)

[Principle of Least Privilege](#)

[Security Managers](#)

[Class Loaders](#)

[Summary](#)

## [Chapter 2 Input Validation and Data Sanitization \(IDS\)](#)

[Rules](#)

[Risk Assessment Summary](#)

[IDS00-J. Sanitize untrusted data passed across a trust boundary](#)

[IDS01-J. Normalize strings before validating them](#)

[IDS02-J. Canonicalize path names before validating them](#)

[IDS03-J. Do not log unsanitized user input](#)

[IDS04-J. Limit the size of files passed to `ZipInputStream`](#)

[IDS05-J. Use a subset of ASCII for file and path names](#)

[IDS06-J. Exclude user input from format strings](#)

[IDS07-J. Do not pass untrusted, unsanitized data to the `Runtime.exec\(\)` method](#)

[IDS08-J. Sanitize untrusted data passed to a regex](#)

[IDS09-J. Do not use locale-dependent methods on locale-dependent data without specifying the appropriate locale](#)

[IDS10-J. Do not split characters between two data structures](#)

[IDS11-J. Eliminate noncharacter code points before validation](#)

[IDS12-J. Perform lossless conversion of String data between differing character encodings](#)

[IDS13-J. Use compatible encodings on both sides of file or network I/O](#)

### **Chapter 3 Declarations and Initialization (DCL)**

[Rules](#)

[Risk Assessment Summary](#)

[DCL00-J. Prevent class initialization cycles](#)

[DCL01-J. Do not reuse public identifiers from the Java Standard Library](#)

[DCL02-J. Declare all enhanced for statement loop variables final](#)

### **Chapter 4 Expressions (EXP)**

[Rules](#)

[Risk Assessment Summary](#)

[EXP00-J. Do not ignore values returned by methods](#)

[EXP01-J. Never dereference null pointers](#)

[EXP02-J. Use the two-argument `Arrays.equals\(\)` method to compare the contents of arrays](#)

[EXP03-J. Do not use the equality operators when comparing values of boxed primitives](#)

[EXP04-J. Ensure that autoboxed values have the intended type](#)

[EXP05-J. Do not write more than once to the same variable within an expression](#)

[EXP06-J. Do not use side-effecting expressions in assertions](#)

### **Chapter 5 Numeric Types and Operations (NUM)**

[Rules](#)

[Risk Assessment Summary](#)

[NUM00-J. Detect or prevent integer overflow](#)

[NUM01-J. Do not perform bitwise and arithmetic operations on the same data](#)

[NUM02-J. Ensure that division and modulo operations do not result in divide-by-zero errors](#)

[NUM03-J. Use integer types that can fully represent the possible range of unsigned data](#)

[NUM04-J. Do not use floating-point numbers if precise computation is required](#)

[NUM05-J. Do not use denormalized numbers](#)

[NUM06-J. Use the `strictfp` modifier for floating-point calculation consistency across platforms](#)

[NUM07-J. Do not attempt comparisons with NaN](#)

- [NUM08-J. Check floating-point inputs for exceptional values](#)
- [NUM09-J. Do not use floating-point variables as loop counters](#)
- [NUM10-J. Do not construct BigDecimal objects from floating-point literals](#)
- [NUM11-J. Do not compare or inspect the string representation of floating-point values](#)
- [NUM12-J. Ensure conversions of numeric types to narrower types do not result in lost or misinterpreted data](#)
- [NUM13-J. Avoid loss of precision when converting primitive integers to floating-point](#)

## **Chapter 6 Object Orientation (OBJ)**

[Rules](#)

[Risk Assessment Summary](#)

- [OBJ00-J. Limit extensibility of classes and methods with invariants to trusted subclasses only](#)
- [OBJ01-J. Declare data members as private and provide accessible wrapper methods](#)
- [OBJ02-J. Preserve dependencies in subclasses when changing superclasses](#)
- [OBJ03-J. Do not mix generic with nongeneric raw types in new code](#)
- [OBJ04-J. Provide mutable classes with copy functionality to safely allow passing instances to untrusted code](#)
- [OBJ05-J. Defensively copy private mutable class members before returning their references](#)
- [OBJ06-J. Defensively copy mutable inputs and mutable internal components](#)
- [OBJ07-J. Sensitive classes must not let themselves be copied](#)
- [OBJ08-J. Do not expose private members of an outer class from within a nested class](#)
- [OBJ09-J. Compare classes and not class names](#)
- [OBJ10-J. Do not use public static nonfinal variables](#)
- [OBJ11-J. Be wary of letting constructors throw exceptions](#)

## **Chapter 7 Methods (MET)**

[Rules](#)

[Risk Assessment Summary](#)

- [MET00-J. Validate method arguments](#)
- [MET01-J. Never use assertions to validate method arguments](#)
- [MET02-J. Do not use deprecated or obsolete classes or methods](#)
- [MET03-J. Methods that perform a security check must be declared private or final](#)
- [MET04-J. Do not increase the accessibility of overridden or hidden methods](#)

[MET05-J. Ensure that constructors do not call overridable methods](#)  
[MET06-J. Do not invoke overridable methods in clone\(\)](#)  
[MET07-J. Never declare a class method that hides a method declared in a superclass or superinterface](#)  
[MET08-J. Ensure objects that are equated are equatable](#)  
[MET09-J. Classes that define an equals\(\) method must also define a hashCode\(\) method](#)  
[MET10-J. Follow the general contract when implementing the compareTo\(\) method](#)  
[MET11-J. Ensure that keys used in comparison operations are immutable](#)  
[MET12-J. Do not use finalizers](#)

## **Chapter 8 Exceptional Behavior (ERR)**

[Rules](#)

[Risk Assessment Summary](#)

[ERR00-J. Do not suppress or ignore checked exceptions](#)  
[ERR01-J. Do not allow exceptions to expose sensitive information](#)  
[ERR02-J. Prevent exceptions while logging data](#)  
[ERR03-J. Restore prior object state on method failure](#)  
[ERR04-J. Do not exit abruptly from a finally block](#)  
[ERR05-J. Do not let checked exceptions escape from a finally block](#)  
[ERR06-J. Do not throw undeclared checked exceptions](#)  
[ERR07-J. Do not throw RuntimeException, Exception, or Throwable](#)  
[ERR08-J. Do not catch NullPointerException or any of its ancestors](#)  
[ERR09-J. Do not allow untrusted code to terminate the JVM](#)

## **Chapter 9 Visibility and Atomicity (VNA)**

[Rules](#)

[Risk Assessment Summary](#)

[VNA00-J. Ensure visibility when accessing shared primitive variables](#)  
[VNA01-J. Ensure visibility of shared references to immutable objects](#)  
[VNA02-J. Ensure that compound operations on shared variables are atomic](#)  
[VNA03-J. Do not assume that a group of calls to independently atomic methods is atomic](#)  
[VNA04-J. Ensure that calls to chained methods are atomic](#)  
[VNA05-J. Ensure atomicity when reading and writing 64-bit values](#)

## **Chapter 10 Locking (LCK)**

[Rules](#)

## Risk Assessment Summary

LCK00-J. Use private final lock objects to synchronize classes that may interact with untrusted code

LCK01-J. Do not synchronize on objects that may be reused

LCK02-J. Do not synchronize on the class object returned by getClass()

LCK03-J. Do not synchronize on the intrinsic locks of high-level concurrency objects

LCK04-J. Do not synchronize on a collection view if the backing collection is accessible

LCK05-J. Synchronize access to static fields that can be modified by untrusted code

LCK06-J. Do not use an instance lock to protect shared static data

LCK07-J. Avoid deadlock by requesting and releasing locks in the same order

LCK08-J. Ensure actively held locks are released on exceptional conditions

LCK09-J. Do not perform operations that can block while holding a lock

LCK10-J. Do not use incorrect forms of the double-checked locking idiom

LCK11-J. Avoid client-side locking when using classes that do not commit to their locking strategy

## **Chapter 11 Thread APIs (THI)**

### Rules

#### Risk Assessment Summary

THI00-J. Do not invoke Thread.run()

THI01-J. Do not invoke ThreadGroup methods

THI02-J. Notify all waiting threads rather than a single thread

THI03-J. Always invoke wait() and await() methods inside a loop

THI04-J. Ensure that threads performing blocking operations can be terminated

THI05-J. Do not use Thread.stop() to terminate threads

## **Chapter 12 Thread Pools (TPS)**

### Rules

#### Risk Assessment Summary

TPS00-J. Use thread pools to enable graceful degradation of service during traffic bursts

TPS01-J. Do not execute interdependent tasks in a bounded thread pool

TPS02-J. Ensure that tasks submitted to a thread pool are interruptible

TPS03-J. Ensure that tasks executing in a thread pool do not fail silently

TPS04-J. Ensure ThreadLocal variables are reinitialized when using thread pools

## **Chapter 13 Thread-Safety Miscellaneous (TSM)**

[Rules](#)

[Risk Assessment Summary](#)

[TSM00-J. Do not override thread-safe methods with methods that are not thread-safe](#)

[TSM01-J. Do not let the `this` reference escape during object construction](#)

[TSM02-J. Do not use background threads during class initialization](#)

[TSM03-J. Do not publish partially initialized objects](#)

## **Chapter 14 Input Output (FIO)**

[Rules](#)

[Risk Assessment Summary](#)

[FIO00-J. Do not operate on files in shared directories](#)

[FIO01-J. Create files with appropriate access permissions](#)

[FIO02-J. Detect and handle file-related errors](#)

[FIO03-J. Remove temporary files before termination](#)

[FIO04-J. Close resources when they are no longer needed](#)

[FIO05-J. Do not expose buffers created using the `wrap\(\)` or `duplicate\(\)` methods to untrusted code](#)

[FIO06-J. Do not create multiple buffered wrappers on a single `InputStream`](#)

[FIO07-J. Do not let external processes block on input and output streams](#)

[FIO08-J. Use an `int` to capture the return value of methods that read a character or byte](#)

[FIO09-J. Do not rely on the `write\(\)` method to output integers outside the range 0 to 255](#)

[FIO10-J. Ensure the array is filled when using `read\(\)` to fill an array](#)

[FIO11-J. Do not attempt to read raw binary data as character data](#)

[FIO12-J. Provide methods to read and write little-endian data](#)

[FIO13-J. Do not log sensitive information outside a trust boundary](#)

[FIO14-J. Perform proper cleanup at program termination](#)

## **Chapter 15 Serialization (SER)**

[Rules](#)

[Risk Assessment Summary](#)

[SER00-J. Maintain serialization compatibility during class evolution](#)

[SER01-J. Do not deviate from the proper signatures of serialization methods](#)

[SER02-J. Sign then seal sensitive objects before sending them across a trust boundary](#)

[SER03-J. Do not serialize unencrypted, sensitive data](#)

[SER04-J. Do not allow serialization and deserialization to bypass the security manager](#)

[SER05-J. Do not serialize instances of inner classes](#)

[SER06-J. Make defensive copies of private mutable components during deserialization](#)

[SER07-J. Do not use the default serialized form for implementation-defined invariants](#)

[SER08-J. Minimize privileges before deserializing from a privileged context](#)

[SER09-J. Do not invoke overridable methods from the `readObject\(\)` method](#)

[SER10-J. Avoid memory and resource leaks during serialization](#)

[SER11-J. Prevent overwriting of externalizable objects](#)

## **Chapter 16 Platform Security (SEC)**

[Rules](#)

[Risk Assessment Summary](#)

[SEC00-J. Do not allow privileged blocks to leak sensitive information across a trust boundary](#)

[SEC01-J. Do not allow tainted variables in privileged blocks](#)

[SEC02-J. Do not base security checks on untrusted sources](#)

[SEC03-J. Do not load trusted classes after allowing untrusted code to load arbitrary classes](#)

[SEC04-J. Protect sensitive operations with security manager checks](#)

[SEC05-J. Do not use reflection to increase accessibility of classes, methods, or fields](#)

[SEC06-J. Do not rely on the default automatic signature verification provided by `URLClassLoader` and `java.util.jar`](#)

[SEC07-J. Call the superclass's `getPermissions\(\)` method when writing a custom class loader](#)

[SEC08-J. Define wrappers around native methods](#)

## **Chapter 17 Runtime Environment (ENV)**

[Rules](#)

[Risk Assessment Summary](#)

[ENV00-J. Do not sign code that performs only unprivileged operations](#)

[ENV01-J. Place all security-sensitive code in a single jar and sign and seal it](#)

[ENV02-J. Do not trust the values of environment variables](#)

[ENV03-J. Do not grant dangerous combinations of permissions](#)

[ENV04-J. Do not disable bytecode verification](#)

[ENV05-J. Do not deploy an application that can be remotely monitored](#)

## **Chapter 18 Miscellaneous (MSC)**

[Rules](#)

[Risk Assessment Summary](#)

[MSC00-J. Use SSLSocket rather than Socket for secure data exchange](#)

[MSC01-J. Do not use an empty infinite loop](#)

[MSC02-J. Generate strong random numbers](#)

[MSC03-J. Never hard code sensitive information](#)

[MSC04-J. Do not leak memory](#)

[MSC05-J. Do not exhaust heap space](#)

[MSC06-J. Do not modify the underlying collection when an iteration is in progress](#)

[MSC07-J. Prevent multiple instantiations of singleton objects](#)

[\*\*Glossary\*\*](#)

[\*\*References\*\*](#)

[\*\*Index\*\*](#)

# Foreword

*James Gosling*

Security in computer systems has been a serious issue for decades. This past decade's explosion in the dependence on networks and the computers connected to them has raised the issue to stratospheric levels. When Java was first designed, dealing with security was a key component. And in the years since then, all of the various standard libraries, frameworks, and containers that have been built have had to deal with security too. In the Java world, security is not viewed as an add-on feature. It is a pervasive way of thinking. Those who forget to think in a secure mindset end up in trouble.

But just because the facilities are there doesn't mean that security is assured automatically. A set of standard practices has evolved over the years. *The CERT<sup>®</sup> Oracle<sup>®</sup> Secure Coding Standard for Java<sup>™</sup>* is a compendium of these practices. These are not theoretical research papers or product marketing blurbs. This is all serious, mission-critical, battle-tested, enterprise-scale stuff.

# Preface

An essential element of secure coding in the Java programming language is a well-documented and enforceable coding standard. The CERT Oracle Secure Coding Standard for Java provides rules for secure coding in the Java programming language. The goal of these rules is to eliminate insecure coding practices that can lead to exploitable vulnerabilities. The application of the secure coding standard leads to higher quality systems that are safe, secure, reliable, dependable, robust, resilient, available, and maintainable and can be used as a metric to evaluate source code for these properties (using manual or automated processes).

This coding standard affects a wide range of software systems developed in the Java programming language.

## Scope

The CERT Oracle Secure Coding Standard for Java focuses on the Java Standard Edition 6 Platform (Java SE 6) environment and includes rules for secure coding using the Java programming language and libraries. *The Java Language Specification*, 3rd edition [JLS 2005] prescribes the behavior of the Java programming language and served as the primary reference for the development of this standard. This coding standard also addresses new features of the Java SE 7 Platform. Primarily, these features provide alternative compliant solutions to secure coding problems that exist in both the Java SE 6 and Java SE 7 platforms.

Languages such as C and C++ allow undefined, unspecified, or implementation-defined behaviors, which can lead to vulnerabilities when a programmer makes incorrect assumptions about the underlying behavior of an API or language construct. The Java Language Specification goes further to standardize language requirements because Java is designed to be a “write once, run anywhere” language. Even then, certain behaviors are left to the discretion of the implementor of the Java Virtual Machine (JVM) or the Java compiler. This standard identifies such language peculiarities and demonstrates secure coding practices to avoid them.

Focusing only on language issues does not translate to writing secure software. Design flaws in Java application programming interfaces (APIs) sometimes lead to their deprecation. At other times, the APIs or the relevant documentation may be interpreted incorrectly by the programming community. This standard identifies such problematic APIs and highlights their correct use. Examples of commonly used faulty design patterns (anti-patterns) and idioms are also included.

The Java language, its core and extension APIs, and the JVM provide security features such as the security manager, access controller, cryptography, automatic memory management, strong type checking, and bytecode verification. These features provide sufficient security for most applications, but their proper use is of paramount importance. This standard highlights the pitfalls and caveats associated with the security architecture and stresses its correct implementation. Adherence to this standard safeguards the confidentiality, integrity, and availability (CIA) of trusted

programs and helps eliminate exploitable security flaws that can result in denial-of-service attacks, time-of-check-to-time-of-use attacks, information leaks, erroneous computations, and privilege escalation.

Software that complies with this standard provides its users the ability to define fine-grained security policies and safely execute trusted mobile code on untrusted systems or untrusted mobile code on trusted systems.

### **Included Libraries**

This secure coding standard addresses security issues primarily applicable to the `lang` and `util` libraries, as well as to the Collections, Concurrency Utilities, Logging, Management, Reflection, Regular Expressions, Zip, I/O, JMX, JNI, Math, Serialization, and XML JAXP libraries. This standard avoids the inclusion of open bugs that have already been fixed or those that lack security ramifications. A functional bug is included only when it is likely that it occurs with high frequency, causes considerable security concerns, or affects most Java technologies that rely on the core platform. This standard is not limited to security issues specific to the Core API but also includes important security concerns pertaining to the standard extension APIs (`javax` package).

### **Issues Not Addressed**

The following issues are not addressed by this standard:

- **Design and Architecture.** This standard assumes that the design and architecture of the product is secure—that is, that the product is free of design-level vulnerabilities that would otherwise compromise its security.
- **Content.** This coding standard does not address concerns specific to only one Java-based platform but applies broadly to all platforms. For example, rules that are applicable to Java Micro Edition (ME) or Java Enterprise Edition (EE) alone and not to Java SE are typically not included. Within Java SE, APIs that deal with the user interface (User Interface Toolkits) or with the web interface for providing features such as sound, graphical rendering, user account access control, session management, authentication, and authorization are beyond the scope of this standard. However, this does not preclude the standard from discussing networked Java systems given the risks associated with improper input validation and injection flaws and suggesting appropriate mitigation strategies.
- **Coding Style.** Coding style issues are subjective; it has proven impossible to develop a consensus on appropriate style rules. Consequently, *The CERT<sup>®</sup> Oracle<sup>®</sup> Secure Coding Standard for Java<sup>™</sup>* recommends only that the user define style rules and apply those rules consistently; requirements that mandate use of any particular coding style are deliberately omitted. The easiest way to consistently apply a coding style is with the use of a code formatting tool. Many integrated development environments (IDEs) provide such capabilities.
- **Tools.** As a federally funded research and development center (FFRDC), the Software Engineering Institute (SEI) is not in a position to recommend particular vendors or tools to enforce the restrictions adopted. Users of this document are

free to choose tools; vendors are encouraged to provide tools to enforce these rules.

- **Controversial Rules.** In general, the CERT secure coding standards try to avoid the inclusion of controversial rules that lack a broad consensus.

## Audience

The *CERT<sup>®</sup> Oracle<sup>®</sup> Secure Coding Standard for Java<sup>™</sup>* is primarily intended for developers of Java language programs. While this standard focuses on the Java Platform SE 6, it should also be informative (although incomplete) for Java developers working with Java ME or Java EE and other Java language versions.

While primarily designed for secure systems, this standard is also useful for achieving other quality attributes such as safety, reliability, dependability, robustness, resiliency, availability, and maintainability.

This standard may also be used by

- Developers of analyzer tools who wish to diagnose insecure or nonconforming Java language programs
- Software development managers, software acquirers, or other software development and acquisition specialists to establish a proscriptive set of secure coding standards
- Educators as a primary or secondary text for software security courses that teach secure coding in Java

The rules in this standard may be extended with organization-specific rules. However, a program must comply with existing rules to be considered conforming to the standard.

Training may be developed to educate software professionals regarding the appropriate application of secure coding standards. After passing an examination, these trained programmers may also be certified as secure coding professionals.

## Contents and Organization

The standard is organized into an introductory chapter and 17 chapters containing rules in specific topic areas. Each of the rule chapters contains a list of rules in that section, and a risk assessment summary for the rules. There is also a common glossary and bibliography. This preface is meant to be read first, followed by the introductory chapter. The rule chapters may be read in any order or used as reference material as appropriate. The rules are loosely organized in each chapter but, in general, may also be read in any order.

Rules have a consistent structure. Each rule has a unique identifier, which is included in the title. The title of the rules and the introductory paragraphs define the conformance requirements. This is typically followed by one or more sets of noncompliant code examples and corresponding compliant solutions. Each rule also includes a risk assessment and bibliographical references specific to that rule. When applicable, rules also list related vulnerabilities and related guidelines from the following sources:

- *The CERT<sup>®</sup> C Secure Coding Standard* [Seacord 2008]
- *The CERT<sup>®</sup> C++ Secure Coding Standard* [CERT 2011]
- ISO/IEC TR 24772. Information Technology—Programming Languages—Guidance to Avoiding Vulnerabilities in Programming Languages through Language Selection and Use [ISO/IEC TR 24772:2010]
- MITRE CWE [MITRE 2011]
- Secure Coding Rules for the Java Programming Language, version 3.0 [SCG 2009]
- *The Elements of Java<sup>™</sup> Style* [Rogue 2000]

## Identifiers

Each rule has a unique identifier, consisting of three parts:

- A three-letter mnemonic, representing the section of the standard, is used to group similar rules and make them easier to find.
- A two-digit numeric value in the range of 00 to 99, which ensures each rule has a unique identifier.
- The letter J, which indicates that this is a Java language rule and is included to prevent ambiguity with similar rules in CERT secure coding standards for other languages.

Identifiers may be used by static analysis tools to reference a particular rule in a diagnostic message or otherwise used as shorthand for the rule title.

## System Qualities

Security is one of many system attributes that must be considered in the selection and application of a coding standard. Other attributes of interest include safety, portability, reliability, availability, maintainability, readability, and performance.

Many of these attributes are interrelated in interesting ways. For example, readability is an attribute of maintainability; both are important for limiting the introduction of defects during maintenance that can result in security flaws or reliability issues. In addition, readability facilitates code inspection by safety officers. Reliability and availability require proper resource management, which also contributes to the safety and security of the system. System attributes such as performance and security are often in conflict, requiring tradeoffs to be made.

The purpose of the secure coding standard is to promote software security. However, because of the relationship between security and other system attributes, the coding standards may include requirements and recommendations that deal primarily with other system attributes that also have a significant impact on security.

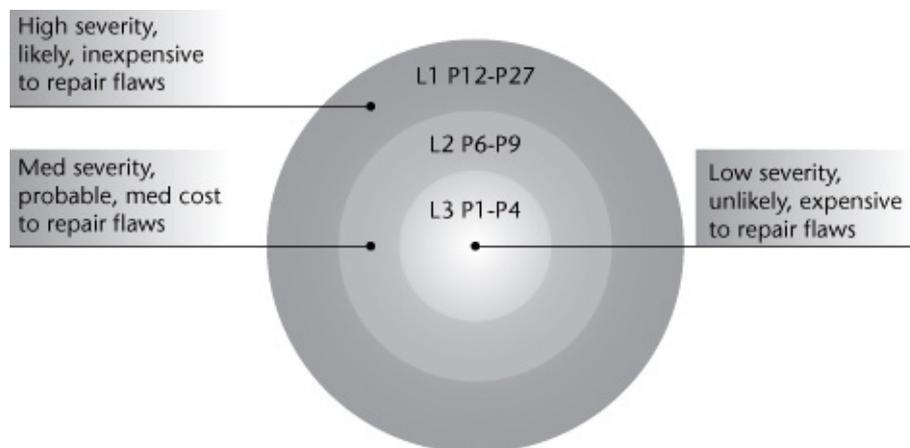
## Priority and Levels

Each rule has an assigned priority. Priorities are assigned using a metric based on Failure Mode, Effects, and Criticality Analysis (FMECA) [IEC 60812]. Three values are assigned for each rule on a scale of 1 to 3 for

- Severity—How serious are the consequences of the rule being ignored:
  - 1 = low (denial-of-service attack, abnormal termination)
  - 2 = medium (data integrity violation, unintentional information disclosure)
  - 3 = high (run arbitrary code, privilege escalation)
- Likelihood—How likely is it that a flaw introduced by violating the rule could lead to an exploitable vulnerability:
  - 1 = unlikely
  - 2 = probable
  - 3 = likely
- Remediation cost—How expensive is it to remediate existing code to comply with the rule:
  - 1 = high (manual detection and correction)
  - 2 = medium (automatic detection and manual correction)
  - 3 = low (automatic detection and correction)

The three values are multiplied together for each rule. This product provides a measure that can be used in prioritizing the application of the rules. These products range from 1 to 27. Rules with a priority in the range of 1 to 4 are level 3 rules, 6 to 9 are level 2, and 12 to 27 are level 1. As a result, it is possible to claim level 1, level 2, or complete compliance (level 3) with a standard by implementing all rules in a level, as shown in [Figure P-1](#).

**Figure P-1. Levels and priority ranges**



The metric is designed primarily for remediation projects and does not apply to new development efforts that are implemented to the standard.

## Conformance Testing

Software systems can be validated as conforming to *The CERT<sup>®</sup> Oracle<sup>®</sup> Secure Coding Standard for Java<sup>™</sup>*.

## Normative vs. Nonnormative Text

Portions of this coding standard are intended to be normative; other portions are