

*The Secure Programming Council's*  
***Essential Skills for Secure Programmers***  
***Using Java/ JavaEE***

*Draft .9, November 19, 2007*

### **Introduction**

Criminal hackers are increasingly targeting web applications, as organizations ranging from TJ Maxx to the New Zealand government are acutely aware. Vulnerabilities in web applications give the attackers direct access to valuable personal information or other sensitive information that can be sold for money or used by national intelligence organizations.

To blunt these attacks, most large organizations are establishing application security initiatives led by managers who have broad responsibility to use tools and training to ensure new and existing applications do not have security flaws, whether built in-house, outsourced, or at commercial software companies.

More than 40 of these organizations are cooperating in establishing standards and metrics that they can use for measuring their application security programs – in essence a “minimum standard of due care” for secure programming. The group, called the Secure Programming Council, has just completed its first consensus document, called “Essential Skills for Secure Programmers Using Java/JavaEE,” and is making the document available for public comment. Once a 60 day comment period has been completed and the suggestions appropriately woven in, the Council will publish the Java Essential Skills document for all to use.

The “Essential Skills” series is designed to enable organizations to ensure that the developers who write their applications can demonstrate that they have mastered secure programming. The council has additional minimum skills standards initiatives under way for C, C++, .NET languages, and PHP and PERL. When combined with an effective secure development life cycle, such skills can be enormously valuable in building more secure applications.

Knowledge and skills are essential only in the context of tasks that programmers must complete, so this document is organized by the security-related tasks that programmers do regularly

The Secure Programming Council has created a set of standardize tests that measure these Essential Skills. These exams can be used in house to find gaps in programmer skills, and for assessing job candidates, consultants, and outsourcing organizations. The tests will be administered in London on December 5, in Washington DC on December 12, and in 15 other cities in the US and Europe over the next eight months. Parallel examinations are also available for on-line administration inside large organizations. Additional data about the tests can be seen at [www.sans.org/gssp](http://www.sans.org/gssp).

### **A Community Effort: Your Input Is Welcome**

The Secure Programming Council Java and JavaEE steering committee is composed of six of the most skilled Java security people in the world. Ed Tracy of Booz Allen & Hamilton, Ryan Berg and Bruce Mayhew of Ounce Labs, Reza Kopaei of Deloitte and Touche, Frank Kim of Kaiser Permanente, and Sherif Koussa of Firsthand Technologies. Assisting the steering committee are a team of equally impressive people: Jeff Williams and Dave Wichers of OWASP, Brook Connor of Morgan Stanley, Sanjay Bahl of Tata Consulting, Justin Schuh of Neohapsis, Dave Grant of Watchfire, Erik Cabetas of Fortify, Jesper Johansson of Amazon, Vincent Liu of Stach & Liu. They all welcome suggestions from interested testers and programmers on the front lines .

Please send comments to [spa@sans.org](mailto:spa@sans.org). Comments received by December 1 will be used in the current version. Comments received later will shape future versions.

**Task 1 – Data Handling** Java programmers must be able to write programs that read input from interfaces, properly validate, and output these data. Programmers should be familiar with these attack scenarios: Cross-site Scripting (XSS) and SQL-Injection.

**01.1.1: Input Validation Principles.** Java programmers must understand that input validation is an important part of building a trust boundary. Consequently input should not be trusted, regardless of its source. Programmers must also understand the white-list and black-list approaches and the tradeoffs between them. Programmers also need to know “when” to validate, not just “what” to validate. Programmers must understand strong “typing,” vs. weak typing, vs. no typing and what that means for Input Validation threats.

**01.1.2: Input Validation Sources.** Java programmers must recognize common sources of input to Java applications. For example, HTTP requests, Applet sockets, serialized streams, configuration files, backend databases, etc.

**01.1.3: Input Validation Techniques.** Java programmers must understand how to validate common data types such as String data as well as uncommon input structures. Familiarity with Regular Expressions, using Struts, Spring, and other tools of Java and Java EE to perform input validation is required.

**01.1.4: Output Encoding.** Java programmers must understand when and how to use output encoding to display data to user interfaces. Besides knowing how to do encode manually, familiarity with technologies like JSP Standard Tag Library (JSTL) and Enterprise Java Beans (EJB) output writing is also required.

**01.1.5: Parameterized Queries.** Java programmers must understand the security risks introduced by using dynamic queries with untrusted data. Programmers must understand how mitigate those risks by using Java’s PreparedStatement to securely interact with databases.

**Task 2 – Authentication & Session Management.** Java applications often require making security decisions based on an identity. End-users or external programs must be able to prove their identity. For the purposes of this examination, Session Management is considered the process of maintaining a remote entity’s authenticated identity for a given period.

Programmers should be familiar with the following attack scenarios: Phishing, eavesdropping, Man-In-The-Middle, and the passerby attack (shoulder-surfing).

**01.2.1: Authenticate Principles.** Programmers must understand authentication principles. For example when to authenticate (trust-boundary), C.A.P.T.C.H.A., multi-factor authentication, and re-authentication.

**01.2.2. Authentication Protection.** Java programmers are required to know how to use encryption and certificates to protect various authentication processes. This includes an understanding of credential strength-of-function, credential expiration, and credential recovery/reset. Java programmers must also understand how to use HTTP and HTML properties to protect Web-based authentication. For example, the Pragma header, the cache controls, and the password input type.

**01.2.3: Session Protection.** For the protection of session tokens, Java programmers are required to understand the implications of several topics including encryption, token strength-of-function, active and inactive timeouts, and re-issuance. Programmers should be familiar with cookie management and cookie protection on the client and server including persistent vs. session cookies, and the following attributes: HTTPOnly, Secure, Domain, and Path. Programmers should be familiar with the following attack scenarios: Cross-Site Request Forgery, Session Fixation, and various methods of

Session Hijacking.

**01.2.4: Authentication Techniques.** Java programmers must be familiar with the more common authentication techniques and APIs available within Java and Java EE technologies. This includes the Java Authentication and Authorization Services (JAAS), JNDI and other backend authentication techniques and principles, and various front-end authentication alternatives such as certificates, forms-based, and Basic and Digest Authentication. This familiarity assumes the programmer will understand the threats and tradeoffs for each technique.

**Task 3 – Access Control (Authorization).** Java applications often require restricting access to assets and functionality based on pre-defined security policies. Active enforcement of these Access Control rules must be ensured by the programmer. An understanding of Access Control in different tiers of an application is necessary. This Task focuses on application-level access control activities. It is not intended to cover code privileges.

**01.3.1: What to Protect.** Resources and Functions. Programmers must be able to actively protect accesses to system data objects (resources) and system functionality (functions). An example resource is a user data object that contains information private to a specific user. An example function is a URI or other calls to privileged functions; user administration, business transactions, etc.

**01.3.2: How to Protect.** Declarative vs. Programmatic. Programmers should understand access schemes that are defined via configuration files (Declarative) and schemes that are defined by active checks in custom code (Programmatic).

**01.3.3: Access Control APIs.** Java developers must understand common Access Control APIs, like the Java Authentication and Authorization Service (JAAS) and Struts' access control.

**Task 4 – Java Types & JVM Management.** Java programmers must understand the security implications of built-in data types and Java-specific memory management.

**01.4.1: java.lang.String.** Java programmers must have a complete mastery of the String class's immutability and how to compare String objects.

**01.4.2: Integer and Double Overflows.** Java programmers must understand the limitations of Java's numerical data types and the resulting security implications.

**01.4.3: Garbage Collector.** Java programmers must have an understanding of how the Java Garbage Collector works and the resulting security implications.

**01.4.4: Class-level Security.** Java programmers must be familiar with accessibility modifiers (*public*, *private*, *protected*, and *default*) and how they can be used to protect classes, class members, and class methods. Programmers should also understand the *final* modifier, class comparisons, *serialization*, *clone*-ability, and inner classes.

**01.4.5: Code Privileges.** Java Programmers must understand how to manage the privileges of code as well as the different protection domains. This includes an understanding of the Security Manager, its policy file, the sandbox, and class-loading.

**01.4.6: Class File Protection.** Java programmers must understand how JAR sealing can be used to protect distributed Java applications. Also familiarity with the concept of code obfuscation is recommended.

**Task 5 – Application Faults & Logging.** All Java application programmers need to be able to properly handle expected and unexpected application faults.

**01.5.1: Exception Handling.** Java application developers must understand Java's try/catch/finally construct to appropriately handle application and system exceptions. Developers must determine how much information should be logged when an exception is encountered depending on the nature of the exception.

**01.5.2: Logging.** Developers must understand the principles behind logging security-relevant events such as login, logoff, credential changes, etc. Developers should also be familiar with java.util.logging, and log4j. Developers must understand what information should and should not be logged in order to ensure non-repudiation, reconstruct an attack, or prevent sensitive information from being logged.

**01.5.3: Configuration of Error Handling.** Java EE developers should be familiar with the configuration to return a default error page for HTTP 404 and 500 errors.

**01.5.4: Fail-safe Connection Patterns.** Java programmers must properly form connection patterns using Java's try/catch/finally to prevent resource leaks and other undesired behavior.

**Task 6 – Encryption Services.** Java programmers must understand when and how to use encryption to protect sensitive data.

**01.6.1: Communications Encryption.** Java application developers must be familiar with the Java Secure Sockets Extension (JSSE) packages as well as how to configure SSL communication for Java EE applications. Developers are also responsible for knowing which of their application's external links should be protected with encryption.

**01.6.2: Encryption of Data at Rest.** Java developers must understand how to store sensitive data in encrypted format. This includes familiarity with common encryption API's and Java random number generation.

**Task 7 – Secure Architecture & Coding Principles.** Programmers must understand architecture-level issues and coding practices that contribute to security.

**01.7.1: Thread Safety.** Java application developers must understand race conditions, deadlock, starvation, and how each can affect system security. Java programmers must also be aware of when it is appropriate to use the Vector vs. the ArrayList. Programmers must be aware of the multi-threading considerations class members, of instance variables, static methods, and Servlet class members. Developers must also understand the risk of data leakage as a result of pooling and caching schemes.

**01.7.2: Singletons.** Java developers must understand when a Singleton is needed and how to implement the Singleton pattern in Java.

**01.7.3: JAVA EE Filters.** Java EE programmers must be familiar with Java EE Filters and how they can be used to implement many of the tasks listed in this document.

**01.7.4: Application Architecture.** Programmers should be familiar with the Model-View-Controller paradigm (MVC) and how it contributes to security. Likewise programmers should be familiar with Inversion of Control (IoC) and how that concept impacts application security.

**01.7.5: Secure Coding Principles.** Programmers should be familiar with technology independent security coding principles. For example: least privilege, secure initialization, separation of data from code (no hard-coding values into programs), type-checking, shallow vs. deep copy, global variables, securely formed *if* and *while* statements (always using comparators and putting constant values first in the comparison), bounds checking, function complexity, off-by-one errors, encapsulation, etc.