2 Rules

Sections

- Rule 00. Input Validation and Data Sanitization (IDS)
- Rule 01. Declarations and Initialization (DCL)
- Rule 02. Expressions (EXP)
- Rule 03. Numeric Types and Operations (NUM)
- Rule 04. Characters and Strings (STR)
- Rule 05. Object Orientation (OBJ)
- Rule 06. Methods (MET)
- Rule 07. Exceptional Behavior (ERR)
- Rule 08. Visibility and Atomicity (VNA)
- Rule 09. Locking (LCK)
- Rule 10. Thread APIs (THI)
- Rule 11. Thread Pools (TPS)
- Rule 12. Thread-Safety Miscellaneous (TSM)
- Rule 13. Input Output (FIO)
- Rule 14. Serialization (SER)
- Rule 15. Platform Security (SEC)
- Rule 16. Runtime Environment (ENV)
- Rule 17. Java Native Interface (JNI)
- Rule 49. Miscellaneous (MSC)
- Rule 50. Android (DRD)

Rule Listing

- DCL00-J. Prevent class initialization cycles
- DCL01-J. Do not reuse public identifiers from the Java Standard Library
- DCL02-J. Do not modify the collection’s elements enhanced for statement
- 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
- ENV06-J. Production code must not contain debugging entry points
- 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 complete 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 throw NullPointerException or any of its ancestors
- ERR09-J. Do not allow untrusted code to terminate the JVM
- EXP00-J. Do not ignore values returned by methods
- EXP01-J. Do not use a null in a case where an object is required
- EXP02-J. Do not use the Object.equals() method to compare two arrays
- EXP03-J. Do not use the equality operators when comparing values of boxed primitives
- EXP04-J. Do not pass arguments to certain Java Collections Framework methods that are a different type than the collection parameter type
- EXP05-J. Do not follow a write by a subsequent write or read of the same object within an expression
- EXP06-J. Expressions used in assertions must not produce side effects
- EXP07-J. Prevent loss of useful data due to weak references
- 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. Release resources when they are no longer needed
- FIO05-J. Do not expose buffers or their backing arrays methods to untrusted code
- FIO06-J. Do not create multiple buffered wrappers on a single byte or character stream
- FIO07-J. Do not let external processes block on IO buffers
- FIO08-J. Distinguish between characters or bytes read from a stream and -1
- 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 convert between strings and bytes without specifying a valid character encoding
- 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
- FIO15-J. Do not reset a servlet's output stream after committing it
- FIO16-J. Canonicalize path names before validating them
- IDS00-J. Prevent SQL injection
- 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. Safely extract files from ZipInputStream
• IDS05-J. Use a safe subset of ASCII for file and path names
• IDS06-J. Exclude unsanitized user input from format strings
• IDS07-J. Sanitize untrusted data passed to the Runtime.exec() method
• IDS08-J. Sanitize untrusted data included in a regular expression
• IDS09-J. Specify an appropriate locale when comparing locale-dependent data
• IDS10-J. Don't form strings containing partial characters
• IDS11-J. Perform any string modifications before validation
• IDS13-J. Use compatible character encodings on both sides of file or network IO
• IDS14-J. Do not trust the contents of hidden form fields
• IDS15-J. Do not allow sensitive information to leak outside a trust boundary
• IDS16-J. Prevent XML Injection
• IDS17-J. Prevent XML External Entity Attacks
• JNI00-J. Define wrappers around native methods
• JNI01-J. Safely invoke standard APIs that perform tasks using the immediate caller's class loader instance (loadLibrary)
• JNI02-J. Do not assume object references are constant or unique
• JNI03-J. Do not use direct pointers to Java objects in JNI code
• JNI04-J. Do not assume that Java strings are null-terminated
• 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 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. Use a correct form of the double-checked locking idiom
• LCK11-J. Avoid client-side locking when using classes that do not commit to their locking strategy
• 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 call overriding methods
• MET06-J. Do not invoke override methods in client code
• MET07-J. Never declare a class method that hides a method declared in a superclass or superinterface
• MET08-J. Preserve the equality contract when overriding the equals() method
• 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
• MET13-J. Do not assume that reassigning method arguments modifies the calling environment
• 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
• MSC08-J. Do not store nonserializable objects as attributes in an HTTP session
• MSC09-J. For OAuth, ensure (a) [relying party receiving user’s ID in last step] is same as (b) [relying party the access token was granted to].
• MSC10-J. Do not use OAuth 2.0 implicit grant (unmodified) for authentication
• MSC11-J. Do not let session information leak within a servlet
• 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 remainder 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
• 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
• NUM14-J. Use shift operators correctly
• OBJ00-J. Limit accessibility of fields
• OBJ01-J. Preserve dependencies in subclasses when changing superclasses
• OBJ02-J. Prevent heap pollution
• OBJ03-J. Provide mutable classes with copy functionality to safely allow passing instances to untrusted code
• OBJ04-J. Do not return references to private mutable class members
• OBJ05-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 fields
• OBJ11-J. Be wary of letting constructors throw exceptions
• OBJ12-J. Respect object-based annotations
• OBJ13-J. Ensure that references to mutable objects are not exposed
• OBJ14-J. Do not use an object that has been freed.

Rule AA. References

Rule BB. Glossary

Rule. Preface

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 Trusted code must discard or clean any arguments provided by untrusted code
• SEC09-J Never leak the results of certain standard API methods from trusted code to untrusted code
• SEC10-J Never permit untrusted code to invoke any API that may (possibly transitively) invoke the reflection APIs
• SER00-J. Enable serialization compatibility during class evolution
• SER01-J. Do not deviate from the proper signatures of serialization methods
• SER02-J. Sign then seal objects before sending them outside 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 classes with 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
• SER12-J. Prevent deserialization of untrusted data
• SER13-J. Deserialization methods should not perform potentially dangerous operations
• STR00-J. Don't form strings containing partial characters from variable-width encodings
• STR01-J. Do not assume that a Java char fully represents a Unicode code point
• STR02-J. Specify an appropriate locale when comparing locale-dependent data
• STR03-J. Do not encode noncharacter data as a string
• STR04-J. Use compatible character encodings when communicating string data between JVMs
• 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
• TSM00-J. Do not override thread-safe methods with methods that are not thread-safe
• TSM01-J. Do not let this reference escape during object construction
• TSM02-J. Do not use background threads during class initialization
• TSM03-J. Do not publish partially initialized objects
• 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