## **Java 8 Security Enhancements**

1) The Java Secure Socket Extension(JSSE) provider enables the protocols Transport Layer Security (TLS) 1.1 and TLS 1.2 by default on the client side.

2) A improved method AccessController.doPrivileged has been added which enables code to assert a subset of its privileges, without preventing the full traversal of the stack to check for other permissions.

3) Advanced Encryption Standard (AES) and Password-Based Encryption (PBE) algorithms, such as PBEWithSHA256AndAES\_128 and PBEWithSHA512AndAES\_256 has been added to the SunJCE provider.

4) Java Secure Socket Extension (SunJSSE) has enabled Server Name Indication (SNI) extension for client applications by default in JDK 7 and JDK 8 supports the SNI extension for server applications. The SNI extension is a feature that extends the SSL/TLS protocols to indicate what server name the client is attempting to connect to during handshaking.

5) The SunJSSE is enhanced to support Authenticated Encryption with Associated Data (AEAD) algorithms. The Java Cryptography Extension (SunJCE) provider is enhanced to support AES/GCM/NoPadding cipher implementation as well as Galois/Counter Mode (GCM) algorithm parameters.

6) A new command flag -importpassword is added to the keytool utility. It is used to accept a password and store it securely as a secret key. Classes such as java.security.DomainLoadStoreParameter andjava.security.PKCS12Attribute is added to support DKS keystore type.

7) In JDK 8, the cryptographic algorithms have been enhanced with the SHA-224 variant of the SHA-2 family of message-digest implementations.

8) Enhanced support for NSA Suite B Cryptography which includes:

* OID registration for NSA Suite B cryptography algorithms
* Support for 2048-bit DSA key pair generation and additional signature algorithms for 2048-bit DSA keys such as SHA224withDSA and SHA256withDSA.
* Lifting of the keysize restriction from 1024 to 2048 for Diffie-Hellman (DH) algorithm.

9) SecureRandom class provides the generation of cryptographically strong random numbers which is used for private or public keys, ciphers and signed messages. The getInstanceStrong() method was introduced in JDK 8, which returns an instance of the strongest SecureRandom. It should be used when you need to create RSA private and public key. SecureRandom includes following other changes:

* Two new implementations has introduced for UNIX platforms, which provide blocking and non-blocking behavior.

10) A new PKIXRevocationChecker class is included which checks the revocation status of certificates with the PKIX algorithm. It supports best effort checking, end-entity certificate checking, and mechanism-specific options.

11) The Public Key Cryptography Standards 11 (PKCS) has been expanded to include 64-bit supports for Windows.

12) Two new rcache types are added to Kerberos 5. Type none means no rcache at all, and type dfl means the DFL style file-based rcache. Also, the acceptor requested subkey is now supported. They are configured using the sun.security.krb5.rcache and sun.security.krb5.acceptor.subkey system properties.

13) In JDK 8, Kerberos 5 protocol transition and constrained delegation are supported within the same realm.

14) Java 8 has disabled weak encryption by default. The DES-related Kerberos 5 encryption types are not supported by default. These encryption types can be enabled by adding allow\_weak\_crypto=true in the krb5.conf file.

15) You can set server name to null to denote an unbound server. It means a client can request for the service using any server name. After a context is established, the server can retrieve the name as a negotiated property with the key name SASL.BOUND\_SERVER\_NAME.

16) Java Native Interface (JNI) bridge to native Java Generic Security Service (JGSS) is now supported on Mac OS X. You can set system property sun.security.jgss.native to true to enable it.

17) A new system property, jdk.tls.ephemeralDHKeySize is defined to customize the ephemeral DH key sizes. The minimum acceptable DH key size is 1024 bits, except for exportable cipher suites or legacy mode (jdk.tls.ephemeralDHKeySize=legacy).

18) Java Secure Socket Extension (JSSE) provider honors the client's cipher suite preference by default. However, the behavior can be changed to respect the server's cipher suite preference by calling SSLParameters.setUseCipherSuitesOrder(true) over the server.

## **Java 8 Tools Enhancements**

1) A jjs command is introduced, which invokes the Nashorn engine either in interactive shell mode, or to interpret script files.

2) The java command is capable of launching JavaFX applications, provided that the JavaFX application is packaged correctly.

3) The java command man page (both nroff and HTML) has been completely reworked. The advanced options are now divided into Runtime, Compiler, Garbage Collection, and Serviceability, according to the area that they affect. Several previously missing options are now described. There is also a section for options that were deprecated or removed since the previous release.

4) New jdeps command-line tool allows the developer to analyze class files to determine package-level or class-level dependencies.

5) You can access diagnostic commands remotely, which were previously accessible only locally via the jcmd tool. Remote access is provided using the Java Management Extensions (JMX), so diagnostic commands are exposed to a platform MBean registered to the platform MBean server. The MBean is the com.sun.management.DiagnosticCommandMBean interface.

6) A new option -tsapolicyid is included in the jarsigner tool which enables you to request a signed time stamp from a Time Stamping Authority and attach it to a signed JAR file.

7) A new method java.lang.reflect.Executable.getParameters is included which allows you to access the names of the formal parameters of any method or constructor. However, .class files do not store formal parameter names by default. To store formal parameter names in a particular .class file, and thus enable the Reflection API to retrieve formal parameter names, compile the source file with the -parameters option of the javac compiler.

8) The type rules for binary comparisons in the Java Language Specification (JLS) Section 15.21 will now be correctly enforced by javac.

9) In this release, the apt tool and its associated API contained in the package com.sun.mirror have been removed.

## **Javadoc Enhancements**

In Java SE 8, the following new APIs were added to the Javadoc tool.

* A new DocTree API introduce a scanner which enables you to traverse source code that is represented by an abstract syntax tree. This extends the Compiler Tree API to provide structured access to the content of javadoc comments.
* The javax.tools package contains classes and interfaces that enable you to invoke the Javadoc tool directly from a Java application, without executing a new process.
* The "Method Summary" section of the generated documentation of a class or interface has been restructured. Method descriptions in this section are grouped by type. By default, all methods are listed. You can click a tab to view methods of a particular type (static, instance, abstract, concrete, or deprecated, if they exist in the class or interface).
* The javadoc tool now has support for checking the content of javadoc comments for issues that could lead to various problems, such as invalid HTML or accessibility issues, in the files that are generated by javadoc. The feature is enabled by default, and can also be controlled by the new -Xdoclint option.

## **Pack200 Enhancements**

The Java class file format has been updated because of JSR 292 which Supports Dynamically Typed Languages on the Java Platform.

The Pack200 engine has been updated to ensure that Java SE 8 class files are compressed effectively. Now, it can recognize constant pool entries and new bytecodes introduced by JSR 292. As a result, compressed files created with this version of the pack200 tool will not be compatible with older versions of the unpack200 tool.

## **Java 8 I/O Enhancements**

In Java 8, there are several improvements to the java.nio.charset.Charset and extended charset implementations. It includes the following:

* A New SelectorProvider which may improve performance or scalability for server. The /dev/poll SelectorProvider continues to be the default. To use the Solaris event port mechanism, run with the system property java.nio.channels.spi.Selector set to the value sun.nio.ch.EventPortSelectorProvider.
* The size of <JDK\_HOME>/jre/lib/charsets.jar file is decreased.
* Performance has been improvement for the java.lang.String(byte[], ∗) constructor and the java.lang.String.getBytes() method.

## **Java 8 Networking Enhancements**

1) A new class java.net.URLPermission has been added. It represents a permission for accessing a resource defined by a given URL.

2) A package jdk.net has been added which contains platform specific socket options and a mechanism for setting these options on all of the standard socket types. The socket options are defined in jdk.net.ExtendedSocketOptions.

3) In class HttpURLConnection, if a security manager is installed, and if a method is called which results in an attempt to open a connection, the caller must possess either a "connect"SocketPermission to the host/port combination of the destination URL or a URLPermission that permits this request.

If automatic redirection is enabled, and this request is redirected to another destination, the caller must also have permission to connect to the redirected host/URL.

**=========================================**

1. **Collection API improvements**

We have already seen forEach() method and Stream API for collections. Some new methods added in Collection API are:

* + Iterator default method forEachRemaining(Consumer action) to perform the given action for each remaining element until all elements have been processed or the action throws an exception.
  + Collection default method removeIf(Predicate filter) to remove all of the elements of this collection that satisfy the given predicate.
  + Collection spliterator() method returning Spliterator instance that can be used to traverse elements sequentially or parallel.
  + Map replaceAll(), compute(), merge() methods.
  + Performance Improvement for HashMap class with Key Collisions

1. **Concurrency API improvements**

Some important concurrent API enhancements are:

* + ConcurrentHashMap compute(), forEach(), forEachEntry(), forEachKey(), forEachValue(), merge(), reduce() and search() methods.
  + CompletableFuture that may be explicitly completed (setting its value and status).
  + Executors newWorkStealingPool() method to create a work-stealing thread pool using all available processors as its target parallelism level.

1. **Java IO improvements**

Some IO improvements known to me are:

* + Files.list(Path dir) that returns a lazily populated Stream, the elements of which are the entries in the directory.
  + Files.lines(Path path) that reads all lines from a file as a Stream.
  + Files.find() that returns a Stream that is lazily populated with Path by searching for files in a file tree rooted at a given starting file.
  + BufferedReader.lines() that return a Stream, the elements of which are lines read from this BufferedReader.

public class Employee {

 private String name;

private Integer age;

private Double salary;

public Employee(String name, Integer age, Double salary) {

this.name = name;

this.age = age;

 this.salary = salary;

 }

 public String toString(){

  return "Employee Name:"+this.name

       +"  Age:"+this.age

       +"  Salary:"+this.salary;

 }

//getters and setters for name, age and salary go here

//standard equals() and hashcode() code go here

}

|  |
| --- |
| public class CollectionRemoveIfExample {    static List<Employee> employeeList = new ArrayList<>();    public static void main(String[] args) {      employeeList.add(new Employee("Tom Jones", 45, 7000.00));      employeeList.add(new Employee("Harry Major", 25, 10000.00));      employeeList.add(new Employee("Ethan Hardy", 65, 8000.00));      employeeList.add(new Employee("Nancy Smith", 22, 12000.00));      employeeList.add(new Employee("Deborah Sprightly", 29, 9000.00)); |

**employeeList.removeIf((Employee emp) -> emp.getAge() > = 30);**

|  |
| --- |
| System.out.println("Employees below the age of 30");      employeeList.forEach(System.out::println);    }  } |