Java

Versions

Installation

Pour installer manuellement une version (si pas disponible dans les dépôts). Après téléchargement de l'archive :

Mise en place

```
tar zxvf <version>.tar.gz
sudo mv <version> /usr/lib/jvm
```

Prise en compte

```
sudo update-alternatives --install /usr/bin/java  java
/usr/lib/jvm/<version>/bin/java  1
sudo update-alternatives --install /usr/bin/javac javac
/usr/lib/jvm/<version>/bin/javac 1
```

Choix de la version

```
sudo update-alternatives --config java
sudo update-alternatives --config javac
```

Configuration

```
sudo update-alternatives --config java
sudo update-alternatives --config javac
```

Interrogation

```
update-alternatives --display java
update-alternatives --display javac
```

Keystore / Truststore

- Les fichiers Keystore et Truststore sont nécessaires dès que l'on veut communiquer de manière sécurisée en SSL/TLS.
- Le format par défaut de ces fichiers est JKS jusqu'à Java 8.
- Depuis Java 9 le format par défaut pour le keystore est PKCS12 qui est un format standardisé alors JKS est spécifique à Java.

Java Keystore

Contenu

Un keystore est un magasin qui contient des clés privées et les certificats associés ainsi que des certificats de serveurs.

Utilisation

Utilisé pour stocker les informations nécessaires à l'identité d'une entité (par exemple, un serveur ou un client) pour la signature, le déchiffrement, etc.
En général on utilise un keystore en tant que serveur pour aller chercher une clef privée et présenter une clef publique correspondante et son certificat au client.

Lister le contenu d'un keystore

```
keytool -v -list -storetype jks -keystore <keystore.jks>
```

Extraire le rootCA d'un keystore

Après avoir identifié l'alias du rootCA à l'aide de la commande précédente :

```
keytool -exportcert -alias <rootca> -file <rootCA.crt> -keystore
<keystore.jks>
```

Vérifier le contenu du rootCA

```
openssl x509 -in <rootCA.crt> -text -noout
```

Java truststore

Contenu

Contient uniquement des certificats de CA, pas de clés privées.

Utilisation

Utilisé pour vérifier les certificats présentés par d'autres entités, assurant que les communications sont sécurisées et de confiance.
Typiquement utilisé par le client pour vérifier le certificat présenté par le serveur.

Lister le contenu d'un trustore

```
keytool -v -list -storetype jks -keystore <fichier_keystore.jks>
```

Validation

Programme

CertificateValidator.java

```
import javax.net.ssl.*;
import java.io.FileInputStream;
import java.io.IOException;
import java.net.URL;
import java.security.KeyStore;
import java.security.cert.Certificate;
import java.security.cert.X509Certificate;
import java.util.Date;

public class CertificateValidator {

    public static void validateUrlWithKeystore(String urlStr, String
keystorePath, String keystorePassword) {
        try {
            // 1. Charger le keystore
            KeyStore keystore = loadKeystore(keystorePath,
keystorePassword);

            // 2. Configurer le contexte SSL
            SSLContext sslContext = configureSSLContext(keystore);

            // 3. Créer et configurer la connexion
            HttpsURLConnection connection =
createSecureConnection(urlStr, sslContext);
```

```
            // 4. Effectuer la validation
            validateConnection(connection);

        } catch (Exception e) {
            System.err.println("Erreur lors de la validation: " +
e.getMessage());
            e.printStackTrace();
        }
    }

    private static KeyStore loadKeystore(String keystorePath, String
keystorePassword) throws Exception {
        System.out.println("Chargement du keystore: " + keystorePath);
        KeyStore keystore = KeyStore.getInstance("JKS");
        try (FileInputStream fis = new FileInputStream(keystorePath)) {
            keystore.load(fis, keystorePassword.toCharArray());
            System.out.println("Keystore chargé avec succès");
            return keystore;
        }
    }

    private static SSLContext configureSSLContext(KeyStore keystore)
throws Exception {
        TrustManagerFactory tmf =
TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm
());
        tmf.init(keystore);

        SSLContext sslContext = SSLContext.getInstance("TLS");
        sslContext.init(null, tmf.getTrustManagers(), null);
        return sslContext;
    }

    private static HttpsURLConnection createSecureConnection(String
urlStr, SSLContext sslContext)
            throws IOException {
        URL url = new URL(urlStr);
        HttpsURLConnection connection = (HttpsURLConnection)
url.openConnection();
        connection.setSSLSocketFactory(sslContext.getSocketFactory());

        // Configuration du HostnameVerifier personnalisé
        connection.setHostnameVerifier((hostname, session) -> {
            System.out.println("\nVérification du hostname: " +
hostname);
            try {
                Certificate[] certs = session.getPeerCertificates();
                printCertificateInfo(certs);
                return true;
            } catch (SSLPeerUnverifiedException e) {
                System.err.println("Erreur lors de la vérification du
```

```
hostname: " + e.getMessage());
                return false;
            }
        });

        return connection;
    }

    private static void validateConnection(HttpsURLConnection
connection) {
        try {
            System.out.println("\nTentative de connexion à: " +
connection.getURL());

            // Effectuer la connexion
            connection.connect();

            // Récupérer et afficher les informations des certificats
            Certificate[] serverCerts =
connection.getServerCertificates();

            System.out.println("\nConnexion établie avec succès!");
            System.out.println("Certificats du serveur:");
            printCertificateInfo(serverCerts);

            // Vérifier le code de réponse HTTP
            int responseCode = connection.getResponseCode();
            System.out.println("\nCode de réponse HTTP: " +
responseCode);

        } catch (SSLHandshakeException e) {
            System.err.println("\nErreur SSL lors de la négociation: "
+ e.getMessage());
            System.err.println("Vérifiez que le certificat est présent
dans le keystore.");
        } catch (Exception e) {
            System.err.println("\nErreur lors de la validation: " +
e.getMessage());
        } finally {
            connection.disconnect();
        }
    }

    private static void printCertificateInfo(Certificate[] certs) {
        for (int i = 0; i < certs.length; i++) {
            if (certs[i] instanceof X509Certificate) {
                X509Certificate cert = (X509Certificate) certs[i];
                System.out.println("\nCertificat " + (i + 1) + ":");
                System.out.println("  Sujet: " +
cert.getSubjectX500Principal());
                System.out.println("  Émetteur: " +
```

```
cert.getIssuerX500Principal());
                System.out.println("  Numéro de série: " +
cert.getSerialNumber());
                System.out.println("  Valide à partir de: " +
cert.getNotBefore());
                System.out.println("  Valide jusqu'au: " +
cert.getNotAfter());

                // Vérifier si le certificat est expiré
                Date now = new Date();
                if (now.after(cert.getNotAfter())) {
                    System.out.println("  ATTENTION: Ce certificat est
expiré!");
                } else if (now.before(cert.getNotBefore())) {
                    System.out.println("  ATTENTION: Ce certificat
n'est pas encore valide!");
                } else {
                    System.out.println("  État: Certificat valide");
                }
            }
        }
    }

    public static void main(String[] args) {
        // Exemple d'utilisation
        if (args.length != 3) {
            System.out.println("Usage: java CertificateValidator <url>
<keystore_path> <keystore_password>");
            System.out.println("Exemple: java CertificateValidator
https://example.com /path/to/keystore.jks password");
            return;
        }

        String url = args[0];
        String keystorePath = args[1];
        String keystorePassword = args[2];

        validateUrlWithKeystore(url, keystorePath, keystorePassword);
    }
}
```

Fonctionnement

- Charge le keystore
- Configure un contexte SSL avec les certificats du keystore
- Établit une connexion HTTPS avec l'URL
- Vérifie la chaîne de certificats
- Affiche les informations détaillées des certificats

Compilation

```
sudo dnf install java-xx-openjdk-devel # redhat
sudo apt install default-jdk           # ubuntu
javac CertificateValidator.java
```

Utilisation

```
java CertificateValidator <url> <keystore> <mot de passe>
```

Haut de page