Ch. 7. Couche de sécurité du transport (TLS)

De Apache OpenOffice Wiki
< FR | Documentation | HSQLDB Guide

Sommaire

Chapitre 7. Couche de sécurité du transport (TLS)

TLS Support (a.k.a. SSL)

Blaine Simpson

HSQLDB Development Group

<blaine.simpson@admc.com>

$Date: 2006/07/27 21:08:21 $

Les instructions consignées dans ce document sont susceptibles de changement à tout moment. En particulier, nous prévoyons de changer la méthode pour fournir un certificat de mot de passe côté serveur.

Spécifications

(Requirements)

Spécifications de prise en charge de TLS (http://fr.wikipedia.org /wiki/Transport_Layer_Security) pour HSQLDB (Hsqldb TLS Support Requirements)

- Sun Java 2.x et supérieur. (C'est probablement possible avec un moteur Java de IBM, mais je ne pense pas que quelqu'un ait essayé d'exécuter

HSQLDB avec TLS dans un environnement Java de IBM, et je sais que personne du groupe de développement HSQLDB n'a documenté comment définir cet environnement).
- Avec Java 2.x ou 3.x, vous devrez installer JSSE. Votre serveur et / ou client démarrera plus lentement que celui des utilisateurs de Java 4.x. Les utilisateurs côté client ne seront pas capables d'utiliser le protocole https: JDBC (Parce que la manipulation du protocole https n'est pas implémentée dans Java JSSE versions 2.x/3.x ; S'il y avait des demandes, nous pourrions l'envisager).
- Un trousseau JKS muni d'une clé privée (JKS keystore), de façon à lancer le serveur.
- Si vous officiez côté serveur, alors vous devez exécuter un serveur ou serveur web HSQLDB. Cela n'a pas d'importance si vous avez de nouvelles instances de bases de données sous-jacentes, et pas plus si vous réalisez une nouvelle configuration de serveur ou encryptez une configuration de serveur existante. (Vous pouvez basculer l'encryptage actif/inactif à volonté).
- Il vous faut un fichier HSQLDB jar qui a été construit avec le JSSE présent. Si vous avez eu votre distribution HSQLDB 1.7.2 chez nous, le problème est réglé car nous l'avons construit avec Java 1.4 (qui contient JSSE). Si vous avez construit votre propre fichier jar avec Java 1.3, assurez vous d'avoir d'abord installé JSSE.

Cryptage (chiffrement) de la connexion JDBC

(Encrypting your JDBC connection)

A ce jour, Seulement un cryptage certifié par le serveur et dans un seul sens, a été testé. (At this time, only 1-way, server-cert encryption is tested.)

Côté client

(Client-Side)

Utilisez simplement l'un des préfixes de protocole suivants :

Préfixes des URLs Hsqldb TLS

- **`jdbc:hsqldb:hsqls://`**
- **`jdbc:hsqldb:https://`**

A ce moment, ce dernier ne fonctionnera que pour les clients utilisant Java 1.4.

Si le serveur auquel vous voulez vous connecter utilise un certificat approuvé par votre magasin de clés certifié par défaut(trousseau / fournisseur de clés par défaut) (your default trust keystores), alors il n'y a rien d'autre à faire. Sinon vous devez demander à Java de "faire confiance" à la certification du serveur. (C'est un peu trop simpliste de dire que si le certificat du serveur a été

acheté, alors tout est réglé ; si quelqu'un "signe son propre" certificat en le signant lui-même ou en utilisant un certificat privé d'autorité de certification (CA), (using a private ca certificate), alors vous aurez besoin de régler ce paramètre sur trust/confiance).

Vous devez d'abord avoir le certificat (seulement sa partie "publique"). Puisque ce certificat est transmis à tous les clients, vous pouvez l'obtenir en écrivant un client Java qui l'écrira dans le fichier (you could obtain it by writing a java client that dumps it to file), ou alors en utilisant la commande openssl s_client. Puisque dans la plupart des cas, si vous voulez faire confiance à un certificat non commercial, vous avez probablement accès au serveur de clé, je vous montrerai un exemple sur la façon d'obtenir ce dont vous avez besoin depuis le serveur JKS.

Vous devez déjà avoir une certification X509 (http://www.esup-portail.org /consortium/espace/SSO_1B/tech/java/java_x509.html) pour votre serveur. Si vous avez un serveur de clé, vous pouvez alors générer une certification X509 comme ceci :

Exemple 7.1. Export d'un certificat depuis le magasin du serveur

```
keytool -export -keystore server.store -alias existing_alias -file server.cer
```

Dans cet exemple, server.cer est le certificat X509 dont vous avez besoin pour la prochaine étape.

Maintenant, vous devez ajouter ce certificat à un site de confiance (to one of the system trust keystores) ou à un de vos propres fournisseurs de clés. Reportez vous à (section Personnaliser les fournisseurs) the Customizing Stores section in JSSERefGuide.html (http://java.sun.com/j2se/1.4.2/docs/guide /security/jsse/JSSERefGuide.html#CustomizingStores) pour déterminer où sont vos magasins de clés de confiance. (where your system trust keystores are.) Vous pouvez mettre des clés privées où vous voulez. La commande suivante ajoutera le certificat à un magasin existant, ou en créera un nouveau si `client.store` n'existe pas.

Exemple 7.2. Ajouter un certificat au magasin client Example 7.2. Adding a certificate to the client keystore

```
keytool -import -trustcacerts -keystore trust.store -alias new_alias -file server.cer
```

Si vous faites un nouveau magasin, vous aimerez probablement partir d'une copie du magasin par défaut du système, que vous pouvez trouver quelque part à partir de la racine de votre dossier Java (typiquement jre/lib/security/cacerts pour JDK, mais j'ai oublié où il est exactement pour JRE).

A moins que votre système d'exploitation puisse empêcher d'autres personnes d'écrire dans vos fichiers, vous ne voulez probablement pas définir un mot de passe dans le magasin de confiance. (on the trust keystore)

Si vous avez ajouté le certificat à un magasin de confiance du système (If you added the cert to a system trust store), alors vous en avez fini. Autrement vous devrez spécifier votre magasin de confiance personnel (your custom trust keystore) à votre programme client. La manière générique de définir le magasin de confiance est de définir la propriété système javax.net.ssl.trustStore à chaque fois que vous exécutez votre programme client. Par exemple :

Exemple 7.3. Spécifier votre propre magasin de confiance à un client JDBC

(Example 7.3. Specifying your own trust store to a JDBC client)

```
java -Djavax.net.ssl.trustStore=/home/blaine/trust.store -jar /path/to/hsqldb.jar dest-urlid
```

Cet exemple exécute le programme SqlTool. SqlTool a une prise en charge intégrée TLS, toutefois pour SqlTool vous pouvez définir les magasins de confiance sur la base de l'urlid, dans le fichier de configuration(de SqlTool).

N.B. Le nom de l'hôte dans l'URL de votre base de données doit correspondre rigoureusement avec le nom commun du certificat du serveur. C'est à dire que si un certificat d'un site est admc.com, vous ne pouvez pas utiliser `jdbc:hsqldb:hsqls://localhost` ou `jdbc:hsqldb:hsqls://www.admc.com:1100` pour vous y connecter.

Si vous voulez plus de détails, voyez JSSERefGuide.html sur le site de Sun (http://java.sun.com/j2se/1.4.2/docs/guide/security/jsse/JSSERefGuide.html), ou dans le sous-dossier docs/guide/security/jsse de votre documentation Java SE.

Coté serveur

(Server-Side)

Procurez vous un magasin JKS contenant une clé privée. Ensuite définissez la propriété système `javax.net.ssl.keyStore` au chemin de ce fichier, et `javax.net.ssl.keyStorePassword` au mot de passe du magasin (et à la clé privée-- Ils doivent être identiques).

Exemple 7.4. Exécuter un serveur Hsqldb avec un cryptage TLS

```
java -Djavax.net.ssl.keyStorePassword=secret  \
    -Djavax.net.ssl.keyStore=/usr/hsqldb/db/db3/server.store  \
    -cp /path/to/hsqldb.jar org.hsqldb.Server
```

(Ceci est une commande simple séparée en 2 lignes en utilisant le caractère de continuation "\" (backslash). Dans cet exemple, j'utilise un fichier `server.properties` et donc je n'ai pas besoin de donner d'arguments pour spécifier les instances de la base de données ou l'adresse de fin du serveur). (the server endpoint)

{{Documentation/FR/Attention|**Attention**

Entrer un mot de passe par la ligne de commande n'est pas sécurisé. Il est vraiment approprié que lorsque des utilisateurs non fiables n'ont pas accès à votre ordinateur

S'il y avait des demandes d'utilisateurs, nous aurions une façon plus sécurisée de fournir le mot de passe avant peu.

JSSE

Si vous utilisez Java 4.x, vous êtes déjà prêt. Utilisateurs de Java 1.x, vous êtes livrés à vous-mêmes (Le JSSE de Sun ne fonctionne pas avec la 1.x). Utilisateurs de Java 2.x et 3.x à suivre...

Allez voir http://java.sun.com/products/jsse/index-103.html. Si vous êtes d'accord avec les termes et remplissez les conditions, téléchargez la version familiale (domestic) ou globale du logiciel JSSE. Tout ce dont vous avez besoin dans la distribution du logiciel sont les trois fichiers jar. Si vous avez installé JDK, déplacez les trois fichiers jar dans le répertoire `$JAVA_HOME/jre/lib/ext`. Si vous avez une installation JRE, alors déplacez les trois fichiers jar dans le dossier `$JAVA_HOME/lib/ext`.

Joliment sans efforts.

Création d'un magasin de clés privées

(Making a Private-key Keystore)

Il y a principalement deux manières de faire. Soit vous utilisez un certificat signé par une autorité de certification, ou vous fabriquez le votre. Une chose que vous avez besoin de connaître dans les deux cas est que le nom commun du certificat doit être exactement le nom de l'hôte que les clients JDBC utiliseront dans l'URL de leur base de données.

Certificat signé CA

(CA-Signed Cert) Voir : Autorité de certification (http://fr.wikipedia.org /wiki/Certificate_Authority)

Je ne vais pas vous raconter comment obtenir un certificat signé CA par SSL. C'est déjà bien documenté en beaucoup d'autres endroits.

Assuming that you have a standard **pem-style** private key certificate, here's how you can use openssl and the program DERImport to get it into a JKS keystore.

Parce que j'ai déjà passé beaucoup de temps sur ce document, je vous donne juste un exemple.

Exemple 7.5. Ajouter une clé privée pem-style dans un magasin JKS.

(Example 7.5. Getting a pem-style private key into a JKS keystore)

```
openssl pkcs8 -topk8 -outform DER -in Xpvk.pem -inform PEM -out Xpvk.pk8 -nocrypt

openssl x509 -in Xcert.pem -out Xcert.der -outform DER

java DERImport new.keystore NEWALIAS Xpvk.pk8 Xcert.der
```

Important :

Assurez vous de définir le mot de passe de la clé exactement comme celui du magasin !

Vous avez évidemment besoin du programme `DERImport.class`. Faites des recherches internet pour trouver `DERImport.java` ou `DERImport.class` et téléchargez le.

S'il devient difficile de se procurer DERImport, je pourrais écrire un programme qui fait la même chose-- Faites le seulement moi savoir.

Certification non-signée CA

(Non-CA-Signed Cert)

Exécutez `man keytool` ou voyez the Creating a Keystore section of JSSERefGuide.html (http://java.sun.com/j2se/1.4.2/docs/guide/security/jsse/JSSERefGuide.html#CreateKeystore).

Démarrage automatique de Server ou WebServer sous UNIX

(Automatic Server or WebServer startup on UNIX)

Si vous êtes sous UNIX et voulez démarrer et stopper automatiquement un serveur ou serveur web s'exécutant avec encryptage, suivez les instructions du chapitre Démarrage rapide sous UNIX, rappelez-vous de rendre le fichier de configuration du script init lisible seulement par l'administrateur, et renseignez les variables `TLS_PASSWORD` et `TLS_KEYSTORE`.

Si vous utilisez un certificat privé de serveur, assurez vous également de définir le chemin du magasin de confiance (trust store filepath) comme exposé dans le fichier de configuration exemple du script init.

Le message "Attention" ci-dessus s'applique encore ici. Le mot de passe sera visible par quiconque ayant un minimum de compétences sous UNIX et

désirant le déchiffrer.
