Programmation réseaux

TP 10
Sécurité: SSL

Avril 2007

Nous avons vu lors du dernier TP comment créer son propre protocole sécurisé en Java, à l'aide de primitives cryptographiques de base. Il est également possible d'utiliser les protocoles directement fournis par Java, et en particulier SSL.

1 Client SSL

Le protocole SSL (Secure Socket Layer) et son successeur TLS permettent de sécuriser des protocoles préexistants. L'exemple le plus connu est HTTPS, qui correspond à l'utilisation de HTTP par dessus SSL. Du point de vue du programmeur réseau, l'utilisation de SSL est quasiment transparent : comme son nom l'indique, SSL fournit une abstraction très similaire à celle des sockets. En Java, on utilise alors les classes SSLSocket et SSLServerSocket qui étendent respectivement les classes Socket et ServerSocket.

Par contre, la création d'une socket SSL est plus délicate que pour une socket normale. Ainsi, pour une session SSL standard :

- lorsqu'un client se connecte, le serveur s'authentifie tout d'abord, via un certificat signé par un tiers de confiance (CA). Ce certificat contient en particulier la clé publique du serveur.
- après vérification, le client fabrique alors une clé aléatoire qui va servir au chiffrement symétrique de la session. Le client l'envoie ensuite au serveur, sous forme crypté à l'aide de la clé publique du serveur.
- le serveur, à l'aide de sa clé privée, est alors le seul à pouvoir connaître cette clé de session.

Cette phase de "poignée de main" est directement prise en charge par Java. Coté client par exemple, la seule chose à faire est d'inclure les lignes :

```
SSLSocketFactory sf = (SSLSocketFactory) SSLSocketFactory.getDefault();
SSLSocket s = (SSLSocket) sf.createSocket(host, port);
```

Exercice 1 – *Client HTTPS*

Transformer le client HTTP du TP 5 en client HTTPS.

2 Client SSL à travers un proxy

Malheureusement, pour tester votre client, il n'est pas possible d'accéder directement aux serveurs HTTPS d'internet depuis nivose. Il faut soit passer par le proxy de l'UFR, soit utiliser un serveur local sans certificats signés par un tiers (voir section suivante).

Le proxy installé sur la machine `fructidor` (port 3128) permet en particulier de relayer une connection SSL vers l'exterieur. Pour cela, il faut établir une connection non-cryptée vers ce proxy, et lui envoyer une requète HTTP telle que :

```
CONNECT www.java.com:443 HTTP/1.0
```

Si le proxy vous répond via un code 200, alors tout traffic ultérieur sur cette socket sera acheminé à www.java.com. On peut dès lors "recycler" cette socket en une socket SSL de la façon suivante :

```
SSLSocketFactory sf = (SSLSocketFactory) SSLSocketFactory.getDefault();
SSLSocket s = (SSLSocket) sf.createSocket(anciennesocket, host, port, true);
```

Exercice 2 – *Client HTTPS et Proxy*

Modifier le client HTTPS de l'exercice précédent pour être en mesure de récuperer le contenu de `https://www.java.com/fr/` à travers le proxy de l'UFR.

3 Client SSL et manipulation de certificats

Si l'on souhaite éviter d'utiliser le proxy, il est parfaitement possible de faire tourner un serveur HTTPS en local. Vous pouvez utiliser par exemple `~letouzey/HttpsServer.class`, qui attend un numéro de port comme argument. Par contre, comme ce serveur n'a pas de certificat signé par un tiers de confiance, la couche SSL de Java va refuser par défaut de s'y connecter. Pour se servir malgré tout de ce serveur, il faut alors entrer manuellement son certificat dans une base de clés acceptées par Java :

```
keytool -import -file ~letouzey/HttpsServer.crt -keystore trustedstore
```

Cette commande crée un trousseau de clés nommé `trustedstore` dans votre compte et y ajoute le certificat du serveur fourni. Il ne reste plus qu'à indiquer à Java de faire confiance aux clés de ce trousseau, via :

```
System.setProperty("javax.net.ssl.trustStore","trustedstore");
System.setProperty("javax.net.ssl.trustStorePasswd","123456");
```

Exercice 3 – *Client HTTPS et serveur non signé*

Récupérer la page secrète diffusée par le serveur `HttpsServer` fourni.

4 Serveur SSL

Exercice 4 – *Serveur HTTPS*

Transformer le serveur HTTP du TP 5 en serveur HTTPS.

Vous aurez besoin pour votre serveur d'une paire de clés publique/privé. La commande suivante génère une telle paire dans le trousseau `mykeystore` (après le choix d'un mot de passe).

```
keytool -genkey -keystore mykeystore
```

Vos utilisateurs auront alors besoin de votre clé publique exportée sous forme de certificat :

```
keytool -export -rfc -file mycertif.crt -keystore mykeystore
```

Enfin, le code Java suivant vous sera utile :

```
// declaration de l'emplacement du trouseau de cles
System.setProperty("javax.net.ssl.keyStore", "mykeystore");
System.setProperty("javax.net.ssl.keyStorePassword", "MOTDEPASSE");

// creation d'une SSLServerSocket
SSLServerSocketFactory sf =
        (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();
SSLServerSocket s = (SSLServerSocket) sf.createServerSocket(port);
```