heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Applications et Services Internet

# Laboratoire SSL avec JSSE

Rapport de laboratoire

Romain de Wolff
Simon Hintermann

IL2008

20 janvier 2008

# Table des matières

# 1 Introduction

Le but de ce laboratoire est de modifier le serveur web créé en Java durant un laboratoire précédant et d'y implémenter le protocole SSL/TLS. Notre serveur web doit se lancer dans deux mode différent : un mode avec authentification du client nécessaire et un autre sans.

# 2 Utilisation du serveur web

Le lancement du serveur s'effectue à l'aide de la ligne de commande. Pour lancer le serveur avec authentification du client sur le port 443 (port par défaut de SSL) il faut utiliser la commande suivante :

```
java WebServer 1 443
```

Le "1" permet de dire que l'on active l'authentification du client. Pour le rendre facultatif, on utilisera un "0" à la place, comme le montre la commande suivante :

```
java WebServer 0 443
```

Notons que pour lancer le serveur sur le port 443 comme montré ci dessus il faut exécuter la commande en tant qu'administrateur.

# 3 Clé publique générée

Voici la clé publique qui nous a été generée par Keytool :

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBujCCASMCAQAwejELMAkGA1UEBhMCQ0gxCzAJBgNVBAgTA1ZEMREwDwYDVQQHEwhMYXVzYW5u
ZTEZMBcGA1UEChMQd3d3LlRBR0FEQVJULmNvbTEcMBoGA1UECxMTUGVyc29uYWwgV2ViIFNlcnZl
cjESMBAGA1UEAxMJMTI3LjAuMC4xMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCjcLrCVl/h
50CuSHjNevhTrRS0bCQ1oCN27c3hTLdDbLVjDNqUJqziTXpowFTUXmM/hrbKwVzM5+I4krwx/6dW
oVVhaGywxkQwN4mQ2rgFvkdm8xIpPKfyVMTLYRQLfd89qLYC8C0SUR3MqzuNRpT7lnla1RB9A6Mg
IIx53mf8UQIDAQABoAAwDQYJKoZIhvcNAQEEBQADgYEAAht/hvwHdT1Qb5ZPe6EmBbMJe6VozqQT
yzaA2q6+4Y+FzuQ0PT7oePyg22e6HTiEtxRIhNGiCXVlceeNxKYFBGoBGVSGOHvauWFGRntErntQ
X7vYKW5XCjHfEpsMwKsj42b4zMFn743IT/LmiC/NsghW3q+UD7AUslld4+XaX68=
-----END NEW CERTIFICATE REQUEST-----
```

# 4 Réponses aux questions

Lors de la connexion sur notre serveur HTTPS à l'aide du navigateur Firefox, le serveur nous affiche une alerte comme le montre la figure 1

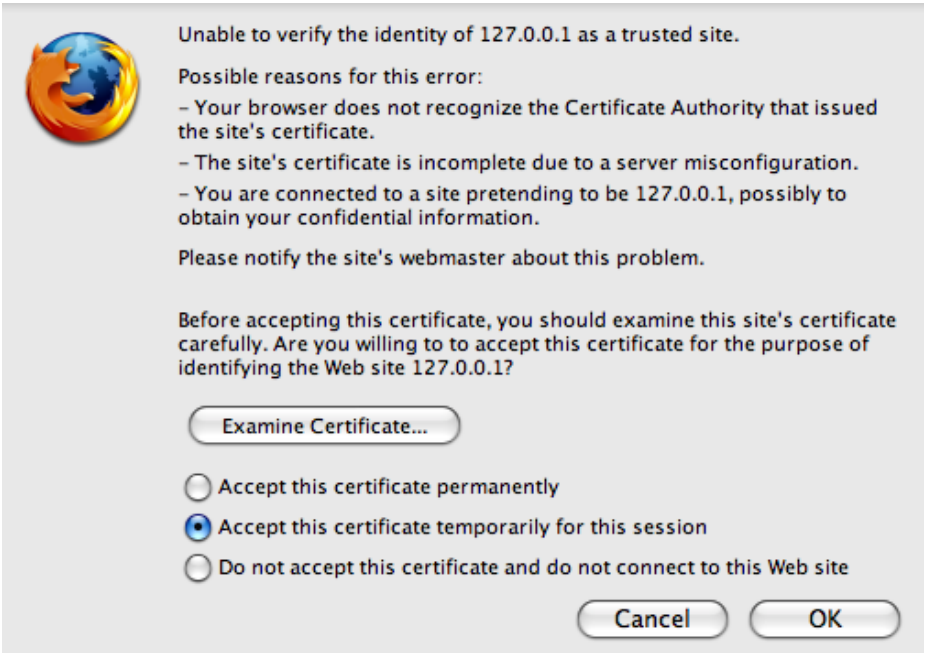


FIG. 1 – Alerte affichée lors de la connexion sur le site sécurisé.

Nous acceptons ce certificat et nous allons voir le site s'afficher. On remarque que le site est sécurisé grâce à l'icone représentant un cadenas (an bas à droite dans Firefox) que l'on peut voire sur la figure 2.

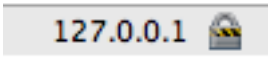


FIG. 2 – Icone dans la barre des tâches du navigateur Firefox.

En cliquant sur le cadenas on peut afficher les informations relatives à la sécurité et donc du certificat que l'on a accepté. La figure 3 nous montre à quoi ressemble cette fenêtre.

La figure 3 nous montre les informations sur le certificat et nous dit que nous faisons confiance au CA mentionné. En cliquant sur le bouton "View" on obtient les informations sur le certificat, exactement les information que nous avons introduites lors de la création à l'aide de *Keytool*. La figure 4 nous montre cette fenêtre.

## 4.1 En examinant le certificat du serveur, comment pouvez-vous en déduire que celui-ci est auto-signé ?

On le voit bien sur la figure 4 : les champs "Issued To" (distribué à) et "Issued By" (distribué par) sont identiques. On sait dès lors que le certificat est auto-signé. Certain navigateur affiche même directement l'information "Certificat


Fig. 3 – Information sur la sécurité du site.

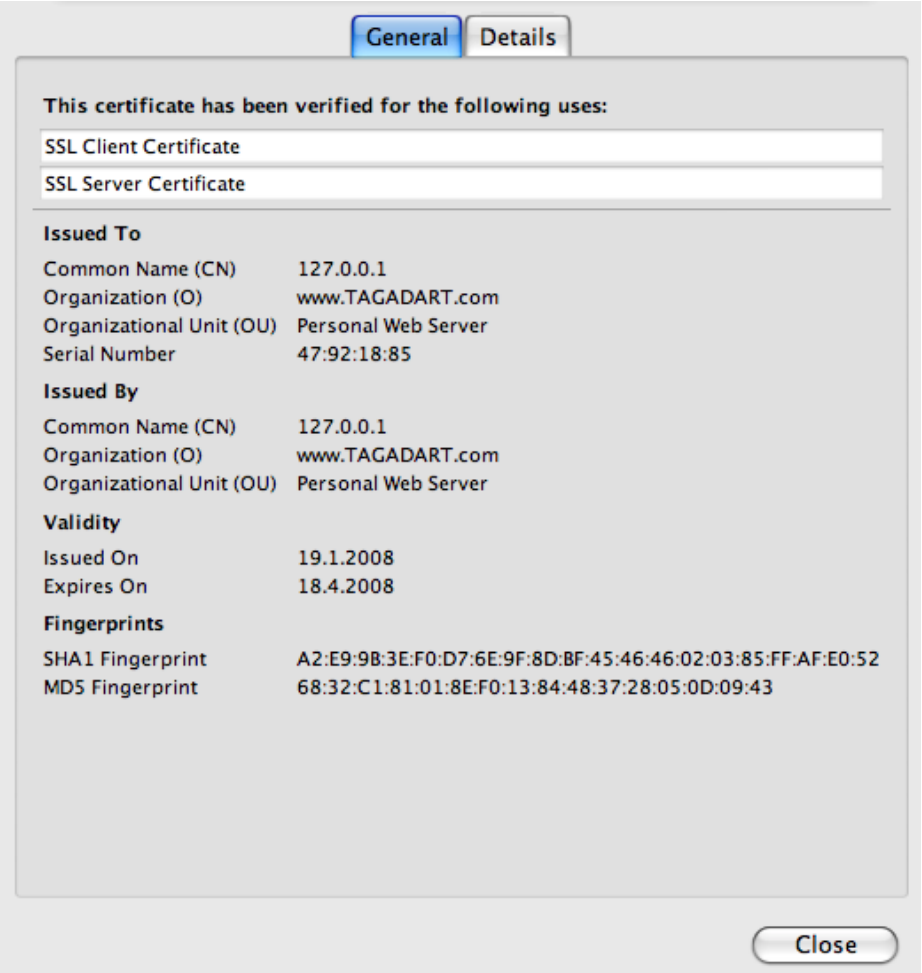


FIG. 4 – Affichage détaillée des informations sur le certificats que nous avons créé.

Auto-Signé". C'est le cas notamment de Safari.

### 4.2 Pourquoi serait-il intéressant de faire signer le certificat par une entité comme Verisign ?

Verisign est une autorité de certification (CA) : elle émet des certificats qu'elle vends. Ces certificats sont réputés fiable. L'avantage d'avoir un certificat d'un CA reconnue est que les utilisateurs qui se connecte sur le site peuvent, grâce à la renomée de Verisign, savoir que le site utilise une encryption de qualité. C'est donc pour des questions de sécurité et de véracité que nous avons avantage à utiliser les services offert par une société comme Verisign.

L'utilisateur sera donc mis en confiance et n'aura plus d'avertissement du navigateur comme quoi le certificat est douteux.

## 5 Observations

Nous allons établir deux connexions avec le serveur web différentes et les comparer. La première sans authentification du client et la seconde avec. L'annexe 1, "Communication SSL" illustre les échanges entre le client et le serveur. Des numéros y figurent auxquelles nous ferons référence lors de nos explications.

Nous allons suivre le déroulement partiel des deux type de connexions. Les captures du logiciel Wireshark sont incluses afin de bien voir ce qu'il se passe lors de la connexion.

Les premiers message échangé (1) font partie du *3 way handshake*. C'est à ce moment que le navigateur web (dans notre exemple Firefox) établit une connexion avec le serveur. Ces messages ne sont pas propres à SSL mais il est intéressant de les mentionner ici.

Le client envoie un message `Client Hello` (2) qui contient la liste des algorithmes et la dernière version de SSL qu'il supporte.

Le serveur effectue les choix de *cipher* ainsi que la version de SSL, la plus récente possible, qu'ils utiliseront pour communiquer.

Le dernier message identique échangé est `Certificate` que le serveur envoie au client. Il est optionnel lorsque l'authentification du client n'est pas nécessaire mais indispensable si l'on désire authentifier le client.

La étapes suivantes changent en fonction du système d'authentification choisit.

### 5.1 Connexion sans l'authentification client

Si l'authentification du client n'est pas nécessaire, le message suivant est `Server Hello Done` (5), envoyé par le serveur. Il permet d'indiquer au client que les négociations initiales sont terminées de son côté.

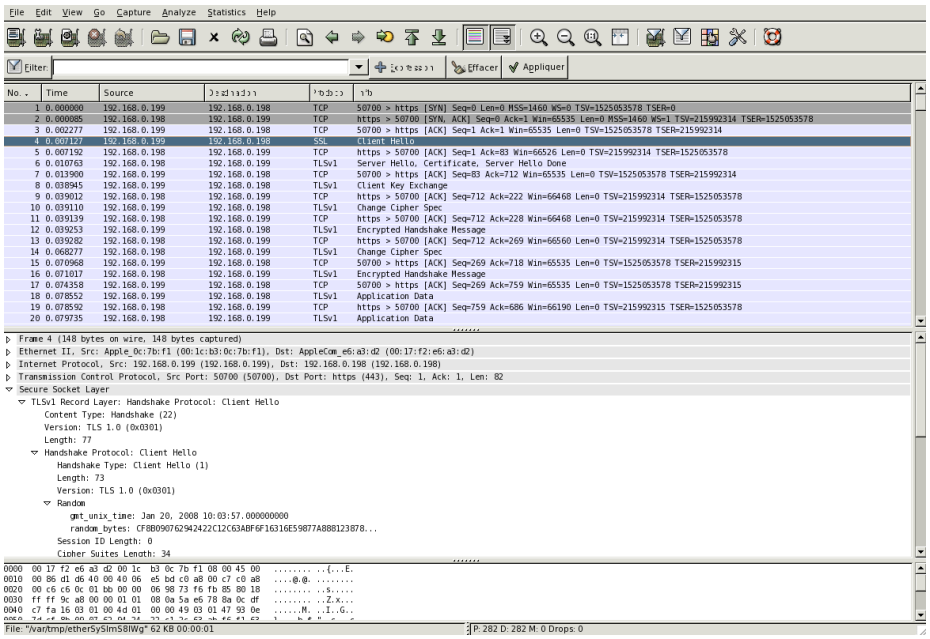


FIG. 5 – Logiciel de capture de paquet Wireshark, communication client-serveur sans authentification du client.

Dans les étapes suivantes, le client va échanger des informations pour permettre de créer les clés symétriques (6, `Client Key Exchange`). Une fois la clé symétrique partagée, le client demande de passer mode sécurisé (7 et 8, `Change Cipher Spec`). Le serveur envoie aussi (8) un message `Change Cipher Spec` qui va permettre finalement de communiquer de manière chiffrée. C'est les message de type `Application Data` qui contiennent les données chiffrées transmises entre le client et le serveur.

## 5.2 Connexion avec l'authentification client activée

Si l'authentification du client est demandée par le serveur, ce dernier va envoyer au client un message `Certificate Request` (4). Le serveur demande au client de lui envoyer son certificat afin de pouvoir décider si il pourra se connecter ou non.

Le client envoie alors son certificat (9) que le serveur peut vérifier.

Les autres message échangé sont identiques que la version qui ne demande pas d'authentification à l'utilisateur. La négociation et le changement pour passer en mode chiffré se font de la même manière.

Dans le cas ou la connexion s'établit correctement, les échanges de messages se déroulent comme illustrés sur la figure 6. Si au contraire, le client est rejeté, la communication sera brutalement interrompue comme illustré sur la figure 7.

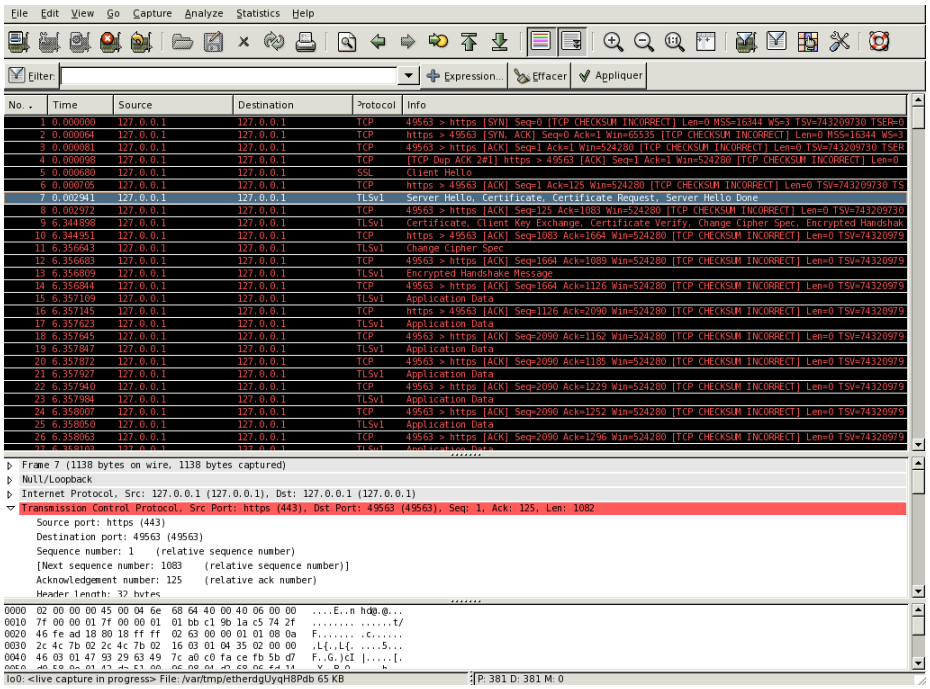


FIG. 6 – Logiciel de capture de paquet Wireshark, communication client-serveur avec authentification du client (connexion OK).

## 5.3 Cipher utilisé

La figure 8 nous montre clairement que le cipher utilisé pour la communication entre le client et le serveur est `TLS_RSA_WITH_RC4_128_MD5`. Le cryptosystème utilisé correspond à celui le plus élevé compatible par le client et le serveur. Le système utilisé variera donc en fonction de ce que supporte le client et le serveur.

Dans la classe `WebServer.java`, nous pouvons changer la liste des *ciphers* utilisé par notre serveur grâce à la constante `CIPHERSUITES`.

```
final static String[] CIPHERSUITES = {
  "SSL_RSA_WITH_3DES_EDE_CBC_SHA",
  "SSL_RSA_WITH_RC4_128_SHA",
  "SSL_RSA_WITH_RC4_128_MD5"
};
```

## 5.4 Principales différences entre SSL v3 et TLS v1

Pour force l'utilisation de l'une ou l'autre des spécifications, on peut modifier l'objet `SSLContext` comme cela :

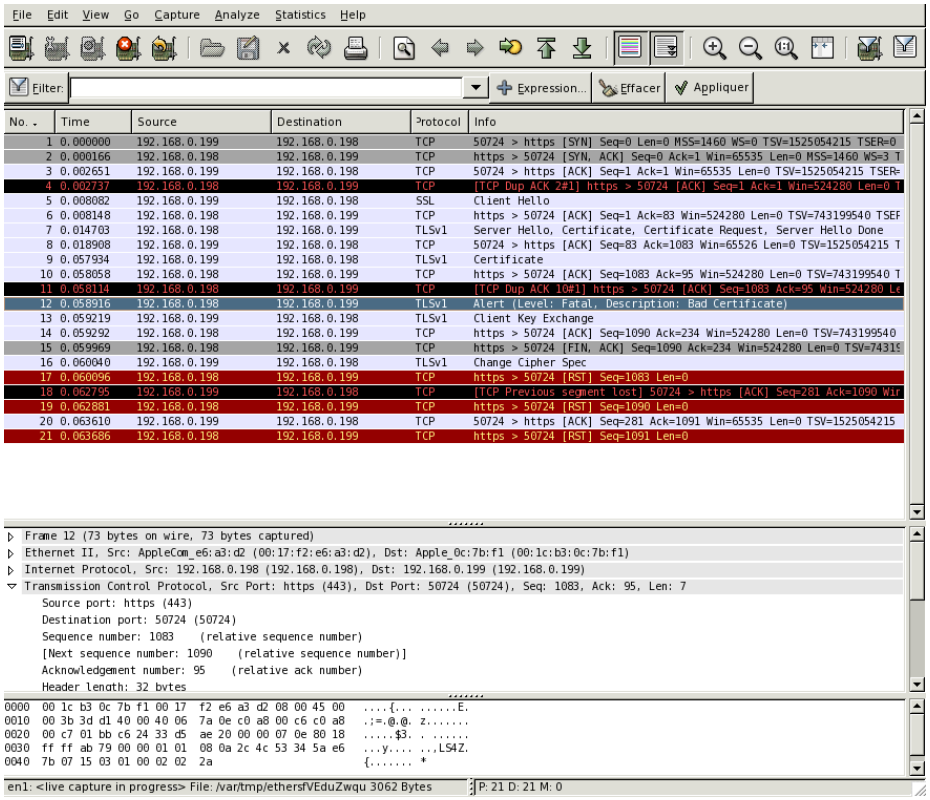


FIG. 7 – Logiciel de capture de paquet Wireshark, communication client-serveur avec authentification du client et erreur (le client ne possède pas le certificat).

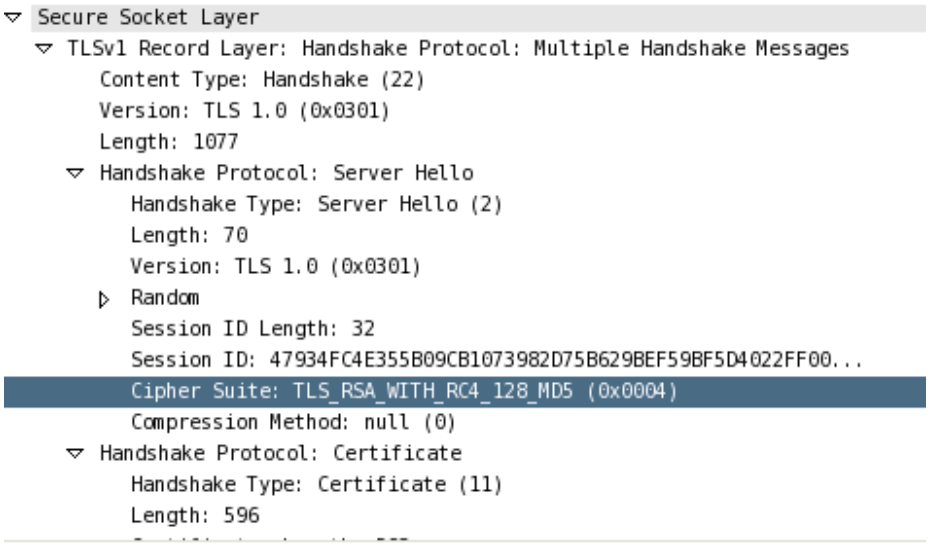


FIG. 8 – Selection du cipher utilisé lors de la communication SSL entre le client et le serveur

```
SSLContext sslContext = SSLContext.getInstance("TLS");
```

Ici on a choisit `TLS`, mais on peut aussi choisir `SSLv3`. On a récupéré ces informations grâce aux méthodes `getEnabledProtocls()` et `getEnabledCipherSuites()`. Nous les affichons dans la console dont voici un extrait :

```
*****************************************
Enabled Session Creation ? false
Protocols actifs:
 - SSLv2Hello
 - SSLv3
 - TLSv1
Ciphers actifs:
 - SSL_RSA_WITH_3DES_EDE_CBC_SHA
 - SSL_RSA_WITH_RC4_128_SHA
 - SSL_RSA_WITH_RC4_128_MD5
*****************************************
```

### 5.5 Le message *Certificate Verifiy*

Le message `Certificate Verify` est utile lorsque l'on desire authentifier le client. Pour se faire, le client signe numériquement à l'aide de sa clé privée ceci afin que le serveur puisse vérifier son identité. Le serveur pourra vérifier en utilisant la clé publique du client.

### 5.6 Configuration du serveur web avec système d'authentification mais sans chiffrement de données

Pour que notre serveur web effectue uniquement de l'authentification, il faut le lancer avec authentification du client et modifier l'objet `serveur` appartenant à la classe `SSLServerSocket`. La méthode `setEnableSessionCreation()` permet d'effectuer cette action.

En effet, si la session n'est pas créer, on aura uniquement l'authentification qui sera effectuée. Le reste des communication ne sera pas chiffrés.

## 6 Conclusion

La découverte et l'implémentation du protocole SSL/TLS est passionnant. Nous n'avions jamais effectué de serveur sécurisé de cette manière. Le langage Java qui met à disposition JSSE nous facilite grandement la vie et rend la création d'un serveur web sécurisé relativement facile.

Après avoir lus la documentation et fait quelques tests, il nous fallut simplement du temps pour faire tout ce qui est demandé dans ce laboratoire.

# 7 Références

**http://java.sun.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html**
Java Secure Socket Extension (JSSE) - Reference Guide, Sun.com

**http://java.sun.com/javase/technologies/security/** Java SE Security, Sun.com

**http://fr.wikipedia.org/wiki/Transport_Layer_Security** TLS et SSL, Wikipedia.org

**http://www.javaworld.com/javatips/jw-javatip115.html** Secure JavaMail with SSL, JavaWorld.com

**http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.itame2.doc_5.1/ss7aumst18.htm**
The SSL Handshake, IBM.com

**http://www.commentcamarche.net/crypto/ssl.php3** Cryptographie SSL, CommentCaMarche.net

**http://www.computing.net/webdevel/wwwboard/forum/439.html** SSL vs TLS, Forum Computing.Net