TP Sécurité réseau – OpenSSL

***

Ayitic – Port au Prince
11 – 16 Août 2014

***

Instructeur: LOISEAU Lucien

Objectifs du TP

Nous avons vu dans le TP précédent qu'il est très facile pour un individu malveillant de dérober des informations qui circulent en clair sur un réseau. Nous avons également vu que le chiffrement tout seul ne suffit pas non plus à toujours assurer la sécurité d'une communication. L'utilsateur étant un maillon souvent faible de la chaîne de sécurité, il est crucial d'être capable de reconnaître qu'une communication est correctement sécurisée. De la même façon, un administrateur doit également être capable de sécuriser ses systèmes d'information. Durant ce TP nous allons donc apprendre à :

- Reconnaître qu'une communication est effectivement bien sécurisée
- Créer un certificat pour un serveur et le faire signer par une autorité de certification (CA)
- Créer un CA et signer des certificats de clients
- Configuration avancée de Apache/SSL

À la fin de cette session, vous serez capable d'utiliser et d'administrer des serveurs de façon sécurisé en utilisant une infrastructure à clé publique (PKI pour Public-Key Infrastructure)

Rappel de cours

Le chiffrement asymétrique

Nous avons déjà vu qu'avec une paire de clés publique/privée il était possible d'avoir des communications qui respectent :

- La confidentialité
- L'authenticité
- L'intégrité

Ce sont autant de principes que l'on cherche à avoir lorsqu'on effectue une transaction bancaire sur Internet par exemple. Dans le TP de cryptographie nous avons vu qu'il suffisait de s'échanger des clés publiques pour communiquer de façon sécurisé. Cependant nous avons également insisté sur le fait qu'il fallait **absolument vérifier** que le propriétaire de la clé publique était bien celui qu'il prétendait être. En effet, si on ne vérifie pas cela, rien n'empêche un attaquant de donner sa clé publique en se faisant passer pour qui il veut.

Le problème de l'échange de clé publique pour le web...

Cet échange préalable de la clé publique est difficile pour les sites web sur Internet car on ne peut pas se procurer manuellement la clé publique de chaque site web avant d'aller le visiter. Comment faire alors pour être absolument certain qu'une clé publique envoyée par un serveur web, soit authentique (et pas une fausse clé publique envoyée par un attaquant se trouvant sur le chemin) ?

Les authorités de certification

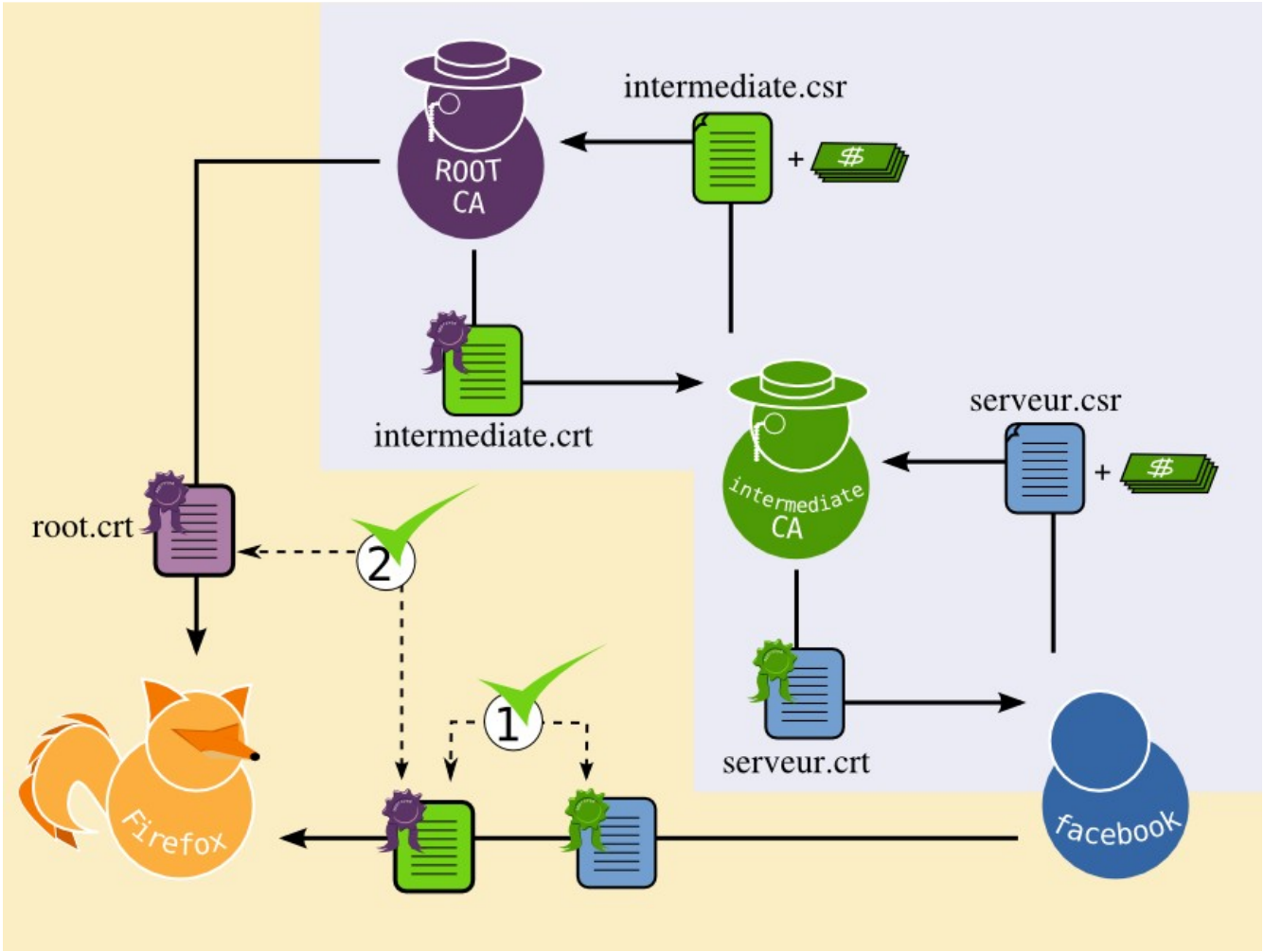
La solution qui a été trouvée repose sur la nécessaire présence d'authorité de certification (**CA**). Un **CA** est une entité de confiance qui possède une paire de clé publique et dont la seule mission est de « signer » d'autres clefs publiques, c'est à dire de les « certifier ». Tout les navigateurs web du monde entier (firefox, chromium, internet explorer, opera, etc.) possède en statique une liste de clé publique de ces authorités de confiances. De cette façon les navigateurs peuvent vérifier qu'un site prétendant avoir été certifié par une autorité de certification le soit effectivement.

Lorsqu'un site web, par exemple facebook.com, crée sa paire de clé publique/privée, le responsable de facebook va aller voir une authorité de certification et lui présenter sa clé publique ainsi qu'une preuve qu'il représente bien la société facebook. Si le CA considère que c'est bon, il va alors pouvoir générer un certificat de confiance contenant la clé publique de facebook et une signature (le hash de la clé publique de facebook signé avec la clé privée du **CA**).

Ainsi, lorsqu'un utilisateur se connectera à facebook, facebook lui donnera fièrement son certificat **facebook.crt**. Comme tout les navigateurs du monde possède la clé publique du CA, le navigateur peut déchiffrer la signature à l'aide de cette clé publique et vérifier que le hash déchiffré corresponde bien au hash de la clé publique contenu dans le certificat donné par facebook.

Les authorités Intermédiaire

La confiance est quelque chose de transitif, c'est à dire qu'il est également possible de faire certifier un certificat d'avoir par des autorités de certification intermédiaire tant que la chaîne de confiance (de certification) est validé. **Pour faire simple** : Le schéma suivant résume le processus de certification du certificat avec une CA intermédiaire :



Pour résumer on a les élements suivants :

- **RootCA** : l'autorité de certification qui signe les certificats, la plus haute autorité (il en existe plusieurs dizaines)
- **IntermediateCA** : Une autorité de certification intermédiaire. Il peut y avoir plusieurs niveau d'autorité intermédiaire à condition que chaque CA soit certifié par une autorité de niveau supérieur jusqu'à arriver à une autorité racine.
- Le **Serveur**: Le serveur édite un certificat qu'il fait signer par un CA. la chaîne de certification doit être entièrement valide jusqu'à la racine.

Vous trouverez un Lexique à la fin de ce TP pour vous aider dans les terminologies
Lorsque vous ne comprenez pas un terme vous pouvez aller voir ce Lexique.
Exemple : PEM, ECDH_RSA,...

Préparation du Banc

Identification du Matériel

Chaque banc est constitué de deux PCs, l'un servira de serveur web hébergeant un site Internet et l'autre un client souhaitant accéder de façon sécurisé à ce service web.

Durant ce TP, nous utiliserons les termes suivants :

- **IPS** : Adresse IP du serveur
- **IPC** : Adresse IP du client
- **IPCA** : Adresse IP du PC de l'instructeur

Afin de simplifier la connexion au serveur, ajoutez une entrée dans le fichier `/etc/hosts` du client de façon à ce que le serveur soit accessible via le nom de domaine « `bancXX-serveur.tp` » en remplaçant XX par votre numéro de banc. Par exemple, si vous êtes le banc numéro 1 vous mettriez « `banc01-serveur.tp` » (utlisez la commande `ip addr show` sur le `serveur` pour voir son adresse IP).

Scenarios

L'instructeur jouera le rôle de l'authorité de certification (Root CA). Vous jouerez le rôle d'un administrateur de site web.

- Dans un premier temps, Vous apprendrez à vérifier que la connexion à un site web est sécurisé ou non

- En tant qu'administrateur, votre première mission sera de créer un certificat puis un « certificate signing request » que vous ferez signer par l'autorité de certification (l'instructeur).

- Puis, vous créerez votre propre autorité de certification et vous déploierez ce RootCA sur tout vos appareils.

- Vous apprendrez à signer un certificat avec votre nouvelle autorité de certification.

- Finalement vous apprendrez à configurer SSL pour assurer une sécurité accrue

Reconnaitre une connexion sécurisée

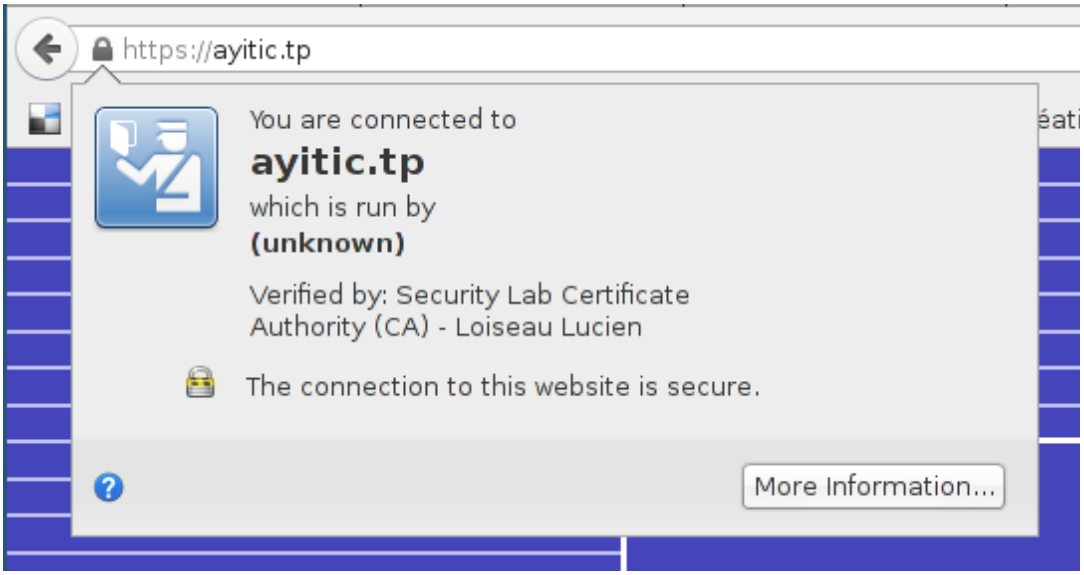
Les certificats certifiés par un CA

Depuis l'ordinateur client, connectez vous à la page suivante (cela nécessite que `ayitic.tp` ait été précédemment ajouté dans le `/etc/hosts`, appelez l'instructeur si ce n'est pas le cas) :

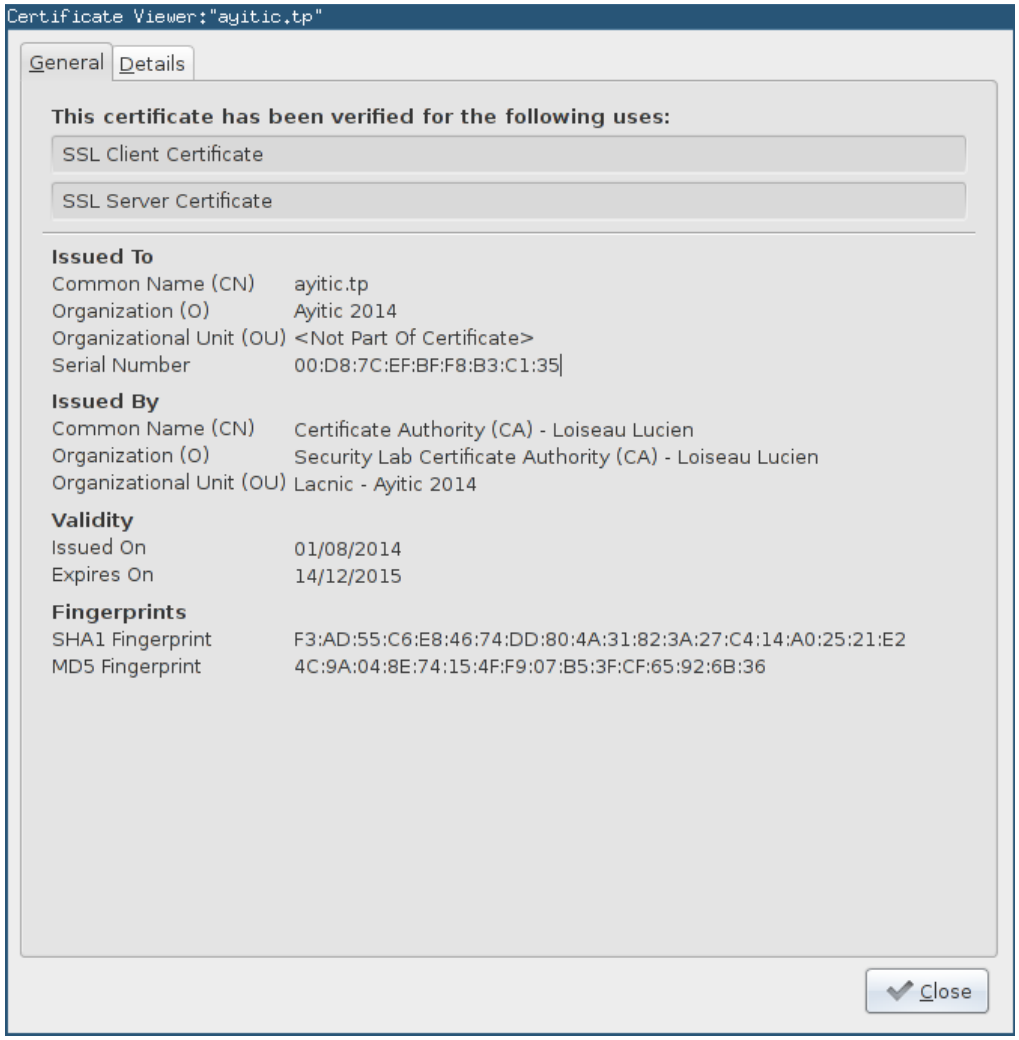
`https://ayitic.tp/`

Vous êtes normalement connecté au site de façon sécurisé comme en témoigne le cadenas gris ainsi que le 's' de https dans la barre d'adresse. Nous allons étudier la chaîne de certification envoyée par le serveur.

Cliquez sur le cadenas dans la barre d'adresse et sur « More Information » puis sur « View Certificate ».



Vous devriez alors avoir accès au détail du certificat comme sur la photo ci-dessous :



Question 1 : Que représente le champs « `Issued To` » ? Et « `Issued By` » ?

Cliquez sur l'onglet « **Détails** » pour accéder au contenu détaillé du certificat. On voit qu'on retrouve les informations précédente « **Issued To** » et « **Issued By** » respectivement dans les champs « **Issuer** » et « **Subject** ». Observez la hiérarchie de certificat ainsi que les différents champs et répondez aux questions suivantes :

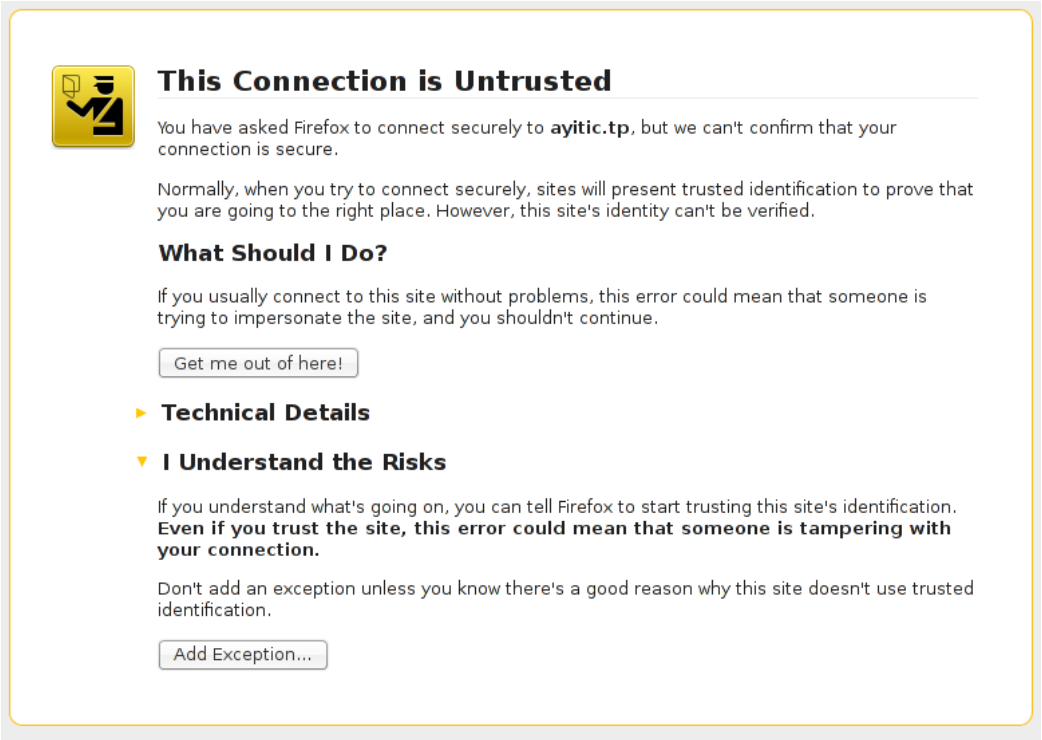
Question 2 : Quel est l'algorithme utilisé pour la clé publique ? Et pour la signature ?

Question 3 : Quelle est la taille de la clé publique ? Est-ce que cela offre une sécurité suffisante ?

Question 4 : Combien de temps ce certificat est-il valide ?

Ce certificat est valide car le certificat du serveur « **ayitic.tp** » est correctement signé par l'autorité de certification « **Certificate Authority (CA) - Loiseau Lucien** » qui se trouve dans le navigateur. De plus, le certificat n'est valide que pour le nom de domaine `ayitic.tp` comme cela est précisé par le champs « **Common Name** » dans la seayitic.tpction « **Subject** » et sera invalide pour tout autre nom de domaine.

Connectez vous à la page sécurisée via l'adresse IP au lieu du nom de domaine `ayitic.tp` cela provoque une erreur dans Firefox :



Question 5 : Dans Technical Detail, quelle est la raison invoquée par l'erreur ?

Les certificats non certifiés par une authorité de certification

Les autorités de certification (CA) résolvent le problème de la distribution préalable des certificats, mais coûtent de l'argent (généralement quelques centaines d'euros par an) et sont plus ou moins cher en fonction de leur politique de vérification. Il est possible de se passer d'une autorité de certification mais comme on va le voir, cela va nécessiter une **vérification manuelle** du certificat puisqu'on ne peut plus compter sur un tiers de confiance pour le faire pour nous.

Attendez que l'instructeur vousayitic.tp informe du changement de certificat et rechargez la page `https://ayitic.tp` (rechargez là avec `Ctrl+Shift+R`). Firefox devrait vous indiquer un message d'erreur.

Question 6 : Dans Technical Detail, quelle est la raison invoqué par l'erreur ? Est-ce la même que précedemment ?

Cette erreur signifie que Firefox n'a pas réussi à valider la chaîne de certification. On peut récupérer le certificat en regardant les détails techniques :

`Add Exception → Get Certificate → View`

Question 7 : Pourrait il s'agir d'un faux certificat envoyé par un attaquant ?

Cliquez sur l'onglet « Détail » afin d'avoir accès aux informations détaillées du certificat. Regardez les différents champs.

*Question 8 : Pourquoi les champs **Issuer** et **Subject** sont ils les même ?*

L'instructeur (et administrateur du site `ayitic.tp`) partage maintenant le hash cryptographique de son certificat afin que vous puissiez procéder à sa vérification manuellement. :

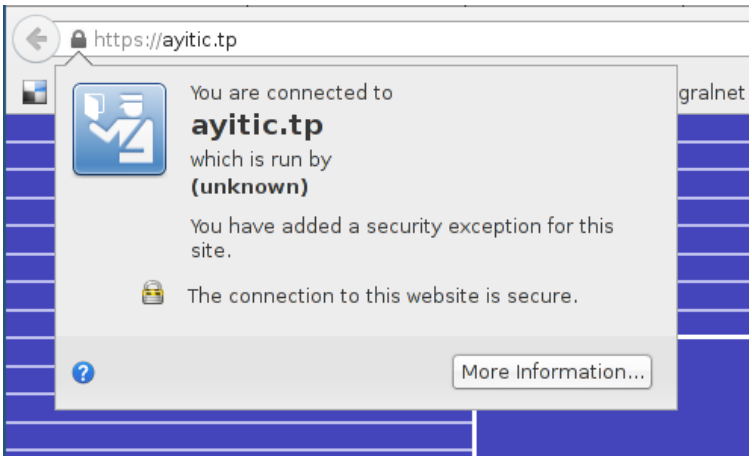
```
SHA1SUM : 4E:10:68:A0:49:2D:58:E5:09:03:B1:B3:50:64:12:32:69:DD:F6:CE
MD5SUM :F7:24:CC:CB:AA:4B:EA:7C:9C:CA:E9:41:5E:2B:39:D0
```

Question 9 : Comparez les hash avec ceux du certificat, le certificat est il authentique ?

Une fois que vous avez vérifié son authenticité et SEULEMENT SI LES HASH CORRESPONDENT, vous pourrez ajoutez une exception dans Firefox afin que cette alerte n'apparaîsse plus à la prochaine connexion au serveur. Pour faire cela, sur la page d'alerte cliquez sur

```
Add Exception → Confirm Security Exception
```

Rechargez la page. Cette fois aucun message d'erreur n'apparait. Vous pouvez toujours avoir accès au certificat en cliquant sur le cadenas dans la barre d'adresse et sur « More Information » puis sur « View Certificate »



Question 10 : Pourquoi faut il vérifier que le certificat est authentique avant d'ajouter une exception ?

Conclusion de la partie I

Dans cette partie nous avons vu comment vérifier que la connexion à un serveur soit correctement sécurisé. L'utilisation d'un tiers de confiance (autorité de certification) nous simplifie grandement la tâche car il n'y a pas de message d'erreur ni de vérification manuelle à effectuer à part bien sûr que nous sommes effectivement connecté en https ! (souvenez vous du TP précédent avec **sslstrip**!)

L'utilisation d'un certificat non certifié est pratique pour sécuriser des connexions à des sites web qui n'ont pas pour ambition d'être déployé pour le grand publique. Ceux-ci nécessitent en revanche impérativement une vérification manuelle du certificat en comparant les empreintes (fingerprint) avant d'ajouter une exception. Cela n'est pas à prendre à la légère car un certificat non certifié peut être émis par un attaquant ! (souvenez vous du TP précédent avec **sslsniff**).

Nous avons terminé du côté utilisateur et allons maintenant apprendre comment générer nous même nos certificats du côté de l'admistrateur web.

Générer et faire certifier un certificat

I. découverte des fichiers de configuration de Apache

Nous allons utiliser l'ordinateur serveur pour configurer et sécuriser Apache. Apache est un serveur HTTP très configurable et l'un des plus utilisé pour héberger les sites web sur Internet. Toute la configuration de Apache se trouve dans le répertoire `/etc/apache2`. Ce repertoire est organisé de la façon suivante :

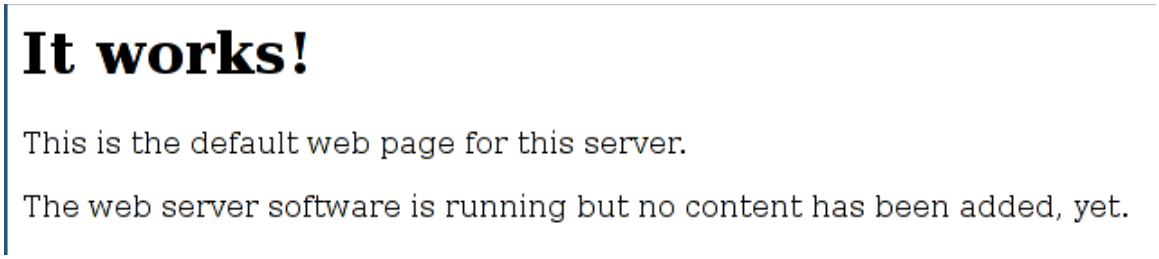
```
/etc/apache2
|-- apache2.conf
|     `-- ports.conf
|-- mods-enabled
|     |-- *.load
|     `-- *.conf
|-- conf.d
|     `-- *
`-- sites-enabled
      `-- *
```

`apache2.conf`	Fichier de configuration principal
`ports.conf`	Appelé par `apache2.conf`, il définit les ports TCP sur lesquels attendre les connexions
`mods-available` `mods-enabled`	Le repertoire `mods-available` contient les fichiers de configuration des modules. Un module est activé en faisant un lien symbolique du fichier de configuration dans le repertoire `mods-enabled`
`sites-available` `sites-enabled`	Un seul serveur HTTP peut héberger différents sites web (appelés hosts). Le repertoire `sites-available` contient les fichiers de configuration des différents hosts. Un hosts est activé en faisant un lien symbolique du fichier de configuration dans le repertoire `sites-enabled`
`conf.d`	Ce répertoire contient des fichiers de configuration fournis par d'autres logiciels ou par l'admnistrateur du système

La configuration par défaut est suffisante et contient même une page par défaut. On démarre le serveur web en lançant le service `apache2` :

```
# /etc/init.d/apache2 start
```

Avec l'ordinateur client, connectez vous à votre serveur web à l'adresse `http://bancXX-serveur.tp`
La page par défaut de apache apparaît alors à l'écran



Question 11 : Où se trouve le fichier de configuration de la page par défaut ?

Regardez ce fichier de configuration, et en particulier la directive `DocumentRoot` qui indique la racine du site web.

Question 12 : Où se trouve la racine du site web (contenant les pages html) ?

On remarque que, bien qu'il y ait un fichier de configuration pour http over ssl (https) dans `/etc/apache2/sites-available/default-ssl`, celui-ci n'est pas activé puisqu'il n'y a pas de lien symbolique dans `/etc/apache2/sites-enabled`. Depuis l'ordinateur client, connectez vous en https au serveur `https://bancXX-serveur.tp`, et vérifiez que cela ne fonctionne pas.

Nous allons éditer notre propre configuration SSL pour apache. Enregistrez la configuration suivante dans le fichier `/etc/apache2/sites-available/default-ssl-ayitic`

~ fichier : /etc/apache2/sites-available/default-ssl-ayitic ~

```
<VirtualHost *:443>
   DocumentRoot /var/www/
   ServerName bancXX-serveur.tp

   SSLEngine on
   SSLCertificateFile      /etc/ssl/private/apache.crt
   SSLCertificateKeyFile   /etc/ssl/private/apache.key


   ErrorLog /var/log/apache2/error.log
   LogLevel warn
   CustomLog /var/log/apache2/access.log combined
</VirtualHost>
```

Comme on peut le voir, ce fichier de configuration va chercher le certificat SSL dans le repertoire `/etc/ssl/private/` qui contiendra la partie publique (apache.crt) et la partie privée (apache.key)

qu'il nous faut généré et signé par l'autorité de certification.

II. Générer un certificat

Aller dans le repertoire `/etc/ssl/private/` et générez une clé privée avec la commande suivante :

```
# openssl genrsa -out apache.key 2048
```

En réalité cette commande génère un couple de clé publique/clé privée, assemblé dans un seul fichier suivant le format **PEM**. Ce fichier `apache.key` doit rester secret et sera utilisé par la configuration du serveur apache (la directive `SSLCertificateKeyFile /etc/ssl/private/apache.key`).

Question 13 : Que se passerait il si quelqu'un se procurait le fichier `apache.key` *?*

Il est possible de protéger cette clé privée avec un mot de passe en ajoutant le paramètre « `-aes128` »

III. Générer un Certificat Signing Request

Une fois que ce fichier est créé, nous allons pouvoir générer un Certificate Signing Request (**csr**) qui est un fichier contenant la clé publique ainsi que certaine informations. Ce fichier **csr** sera ensuite envoyé à une autorité de certification pour signature. Générez le **csr** de la façon suivante :

```
# openssl req -new -key apache.key -out apache.csr
```

ATTENTION : La réponse la plus importante est le champs « **Common Name** » que vous devez répondre par `bancXX-serveur.tp` Répondez soigneusement aux autres questions, c'est ce qui apparaitra sous Firefox dans la section « **Subject** » du certificat.

Il est maintenant temps d'envoyer votre Certificate Signing Request `apache.csr` à l'autorité de certification (l'instructeur) pour signature.

Envoyez apache.csr à l'instructeur, celui-ci vous retournera le certificat signé apache.crt

Question 14 : Que se passerait il pour le CA si vous arriviez à lui faire signer un faux certificat ?

IV. Terminer la configuration du serveur

Placez votre certificat signé `apache.crt` dans le répertoire `/etc/ssl/private/`. Maintenant que nos certificats sont prêt, la dernière chose à faire maintenant est d'activer le SSL sur le serveur web. Pour cela nous allons charger le module SSL et charger notre configuration du site web en SSL. Tapez les commandes suivantes :

```
# a2enmod ssl
# a2ensite default-ssl-ayitic
# /etc/init.d/apache2 restart
```

Explication :

- `a2enmod` : cette commande active un module apache (d'où a2enmod = **a**pache**2 en**able **mod**ule). Ce programme crée simplement un lien symbolique dans /etc/apache2/mods-enabled/. À l'inverse, le programme `a2dismod` permet de désactiver un module (**a**pache**2 dis**able **mod**ule)
- `a2ensite` : De façon similaire cette commande permet d'activer un site en créant un lien symbolique dans le répertoire /etc/apache2/sites-available/
- `/etc/init.d/` : ce répertoire contient les script permettant d'allumer, d'éteindre ou de redémarrer les « démons » du système

Si tout fonctionne bien, vous pouvez maintenant vous connecter depuis le client sur votre site web en https à l'adresse `https://bancXX-serveur.tp`

Appelez l'instructeur pour que celui-ci valide cette étape

Dans la pratique, faire signer un certificat par une autorité de certification est un processus qui peut être coûteux. Comme nous l'avons vu, en utilisant un tiers de confiance, c'est à dire un CA, cela nous simplifie grandement le processus de « vérification » que la clé publique reçu est bien la bonne. Cependant un CA n'est pas obligatoire pour assurer la sécurité d'un site web. Si un site n'a pas vocation à être ouvert au public ou bien ne justifie pas une telle dépense (site perso, site d'entreprise, etc.) il est ainsi possible de se passer du CA en utilisant une des deux solutions suivantes :

- Utiliser un certificat auto-signé
- Créer sa propre autorité de confiance (CA) et signer ses certificats

Dans le cas du certificat auto-signé, Firefox enverra un message d'erreur à la première connexion comme nous l'avons vu au début de ce TP, il faudra alors impérativement vérifier que le certificat est authentique avant de l'ajouter. L'inconvénient de cette méthode c'est qu'il faut créer un certificat par site ce qui signifie pour Firefox d'ajouter une exception par certificat.

Dans le cas où on crée sa propre autorité de certification (CA), il faudra alors prendre soin d'ajouter manuellement ce nouveau CA dans la liste des autorités de confiance de Firefox. L'avantage c'est qu'on peut ensuite signer plusieurs certificats différents avec cette autorité sans avoir à ajouter d'exception pour firefox.

Générer un certificat auto-signé

Générer un certificat auto-signer est extrêment simple puisque cette opération ne requert aucun aller-retour avec un tiers. Allez dans le repertoire `/etc/ssl/private` et tapez la commande suivante :

```
# openssl req -new -x509 -keyout apache-self.key -out apache-self.crt -days 365 -nodes
```

Répondez aux questions et n'oubliez pas que le champs « **Common Name** » doit être le nom de domaine de votre serveur web à savoir `bancXX-serveur.tp`

Modifiez le fichier de configuration de Apache /etc/apache2/sites-available/default-ssl-ayitic pour prendre en compte ce nouveau certificat.

Vous pouvez obtenir le hash de votre certificat sur le serveur en tapant la commande suivante :

```
# openssl x509 -in apache-self.crt -outform der |sha1sum
```

Rechargez la page sous Firefox sur l'ordinateur client en tapant **Ctrl+Shift+R**

Comparez le hash de votre certificat sur le serveur avec celui du certificat sous Firefox, Il doit être le même !

Question 15 : Que se passera t'il quand le serveur changera de certificat à la fin de la période de validité (365 jours) ?

Le certificat auto-signé est pratique pour effectuer des tests ou pour protéger un petit site web mais il devient vite pénible de devoir vérifier à la main les fingerprints. Pour cette raison il est préférable d'évoluer vers sa propre autorité de certification si on veut se passer d'une « vraie » autorité de certification.

Créer sa propre autorité de certification et signer des certificats

I. Créer la clé root de l'autorité de certification (CA Root Key)

Une autorité de certification consiste en réalité à créer une simple paire de clé publique/clé privée. Cela se fait de la même façon que pour la création d'un certificat normal. Créez un répertoire dans `/etc/ssl/authority`, placez vous dedans et tapez la commande suivante pour créer la clé privée :

```
# mkdir /etc/ssl/authority
# cd /etc/ssl/authority
# openssl genrsa -out rootCA.key 2048
```

Important : Gardez cette clé très, très, secrète ! C'est la base de la confiance pour l'ensemble des certificats. Si quelqu'un met la main dessus, il pourra signer n'importe quel certificat et votre navigateur l'acceptera.

II. Auto-Signez la clé pour créer le certificat root

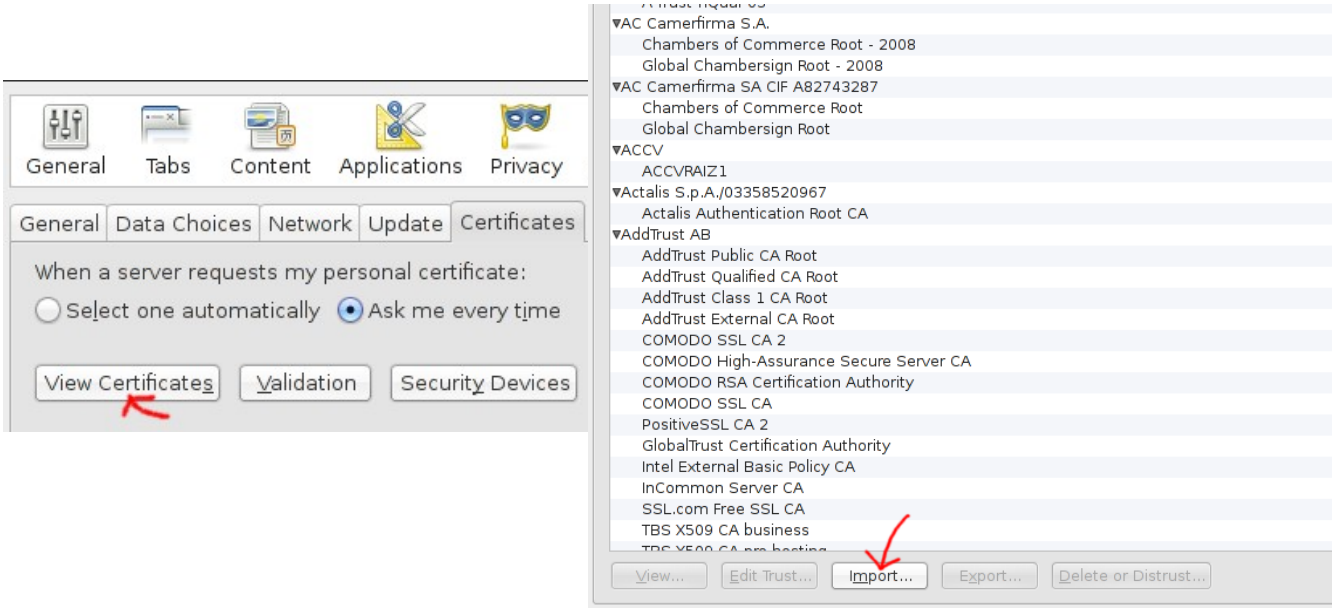
Tapez la commande suivante pour créer le certificat auto-signé :

```
# openssl req -x509 -new -nodes -key rootCA.key -days 1024 -out rootCA.pem
```

Répondez aux questions comme vous le souhaitez, mettez juste votre numéro de banc à un endroit (par exemple répondez à Common Name par « BancXX » en remplaçant XX par votre numéro de banc). Vous obtenez le certificat rootCA.pem, distribuez le à tout les navigateurs du monde ! C'est le certificat publique de votre autorité de certification, avec ça dans sa liste de CA, Firefox pourra vérifier un certificat signé avec ce CA.

On peut ajouter un certificat dans le navigateur Firefox à travers les préférences :

```
Edit → Preferences → Advanced → Certificates → View Certificates → Authorities →  Import
```



Ajoutez RootCA.pem dans le navigateur de votre client
Envoyez RootCA.pem à l'instructeur pour qu'il l'ajoute à son navigateur (par exemple en le plaçant dans le répertoire de votre serveur web `/var/www/`)

Vous possédez maintenant une autorité de certification et vous avez distribué son certificat sur plusieurs navigateurs. Si on se place dans un scénario d'entreprise, ce certificat serait distribué à tout les navigateur des employés. Plus généralement, il faut distribué ce certificat à toutes les personnes susceptible de se connecter à vos services

III Créer un certificat (opération à faire pour chaque site web)

Nous avons déjà généré un certificat pour notre serveur web précédemment. À ce moment là, nous avions généré un **Certificate Signing Request** que nous avions soumis à l'instructeur qui a remis en échange le certificat signé. Nous allons donc reprendre le **Certificate Signing Request** que nous avions déjà généré `/etc/ssl/private/apache.csr` et nous allons le signer avec notre nouvelle autorité de certification.

Créez le repertoire `/etc/ssl/authority/clients` et copiez-y le **certificate signing request** :

```
# mkdir /etc/ssl/authority/clients
# cp /etc/ssl/private/apache.csr /etc/ssl/authority/clients
```

Placez vous dans le dossier authority et signez le certificat de la façon suivante :

```
# cd /etc/ssl/authority
# openssl x509 -req -in clients/apache.csr -CA rootCA.pem -CAkey rootCA.key
-CAcreateserial -out clients/apache.crt -days 500
```

Votre nouveau certificat, signé par votre autorité de certification est maintenant disponible dans le répertoire `clients`. Copiez le dans le repertoire `/etc/ssl/private/` attendu par apache et redémarrez votre serveur apache. Rechargez la page sur l'ordinateur du client avec **Ctrl+Shift+R** et analysez votre nouveau certificat.

Appelez l'instructeur pour qu'il valide cet exercice

Nous avons beaucoup utilisé la ligne de commande openssl dans ce TP. Je vous invite à jeter un œil au document « openssl-cookbook.pdf » situé dans le répertoire de documentation dans la clé qui vous a été fournis qui vous donnera plus de détails sur les possibilités de ce programme.

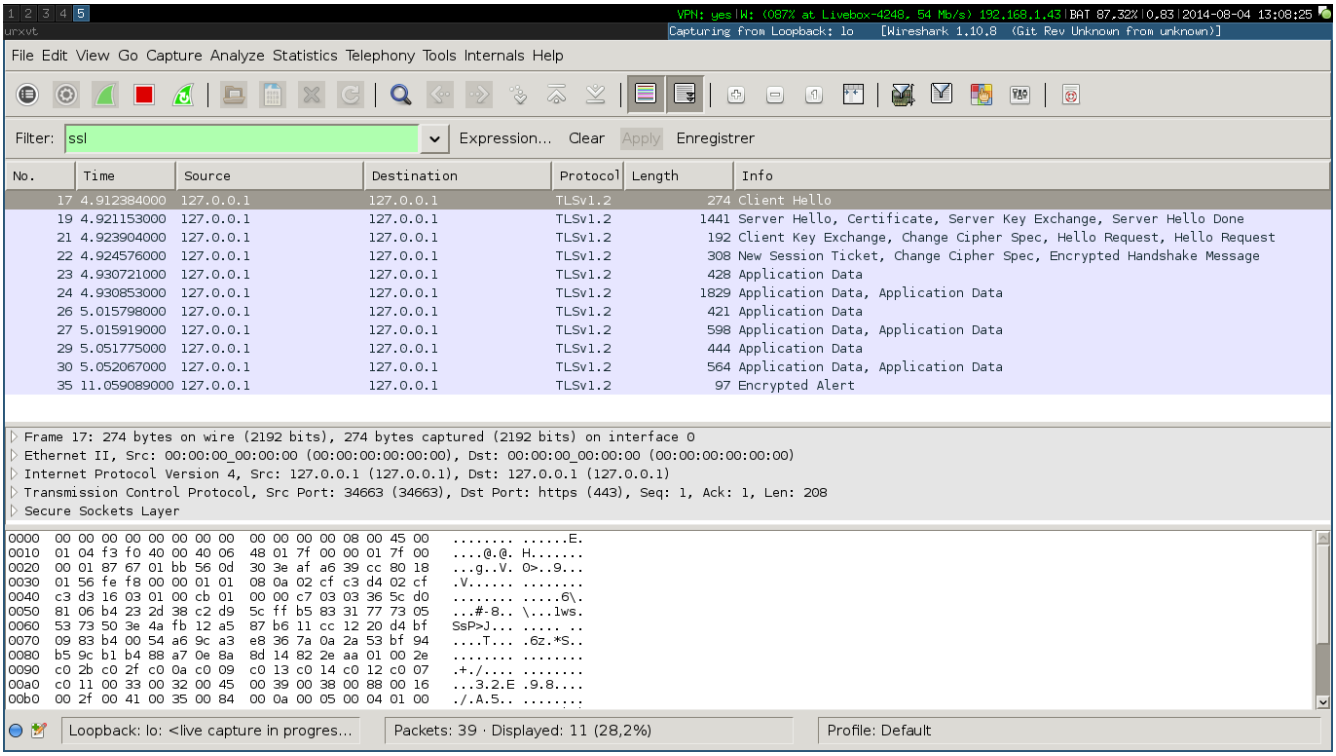
Étude de SSL/TLS et configuration avancée

I. Contexte

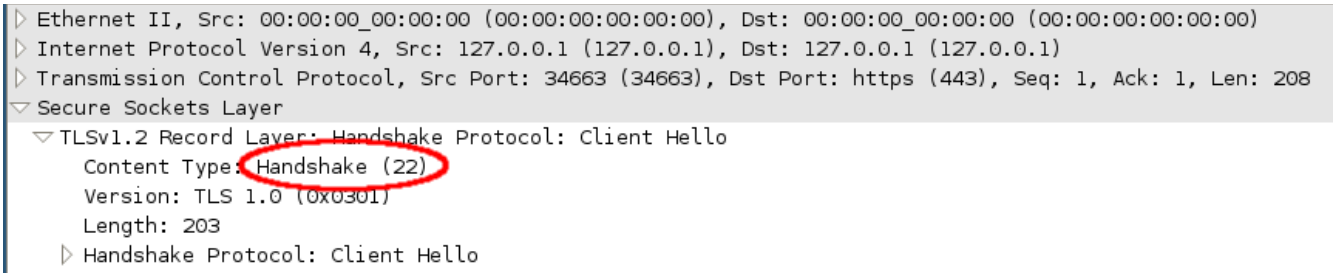
SSL est le premier protocole de sécurisation des échanges sur Internet et fut développé par Netscape jusqu'en 2001 (SSL version 2 et version 3). Le développement et la spécification de SSL est ensuite passé dans les mains de l'IETF (Internet Engineering Task Force) qui a renommé le projet en TLS (Transport Layer Security) pour lequel il existe trois versions, TLSv1.0 (qui est essentiellement SSLv3) TLSv1.1 et TLSv1.2 On parle généralement de SSL pour désigner indifféremment SSL/TLS.

II. Étude de SSL/TLS

Ouvrez wireshark sur l'ordinateur du serveur et démarrez sur une trace, filtrez de façon à n'afficher que le SSL. Sur l'ordinateur du client, connectez vous au serveur en https et observez les traces. Vous devriez obtenir un échange similaire à celui-ci :



TLS enveloppe tout le trafic dans des « enregistrements » de différents types. On peut voir que les premiers octets envoyé par le navigateur est l'octet `0x16 = 22` qui signifie que cet enregistrement est de type « handshake » c'est à dire poignée de main (établissement de la session).



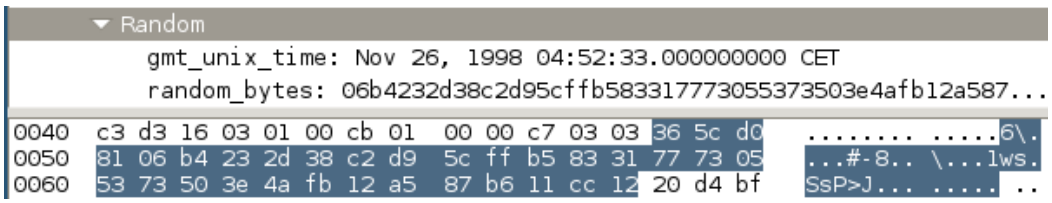
Les deux octets suivants, `0x0301` indique que nous utilisons la version TLSv1.0 (qui est essentiellement SSL v3.1 d'où `0x0301`).

Question 16 : Expliquez de façon schématique les différentes étapes d'une connexion SSL (sans rentrer dans les détails des paquets)

III. Client Hello

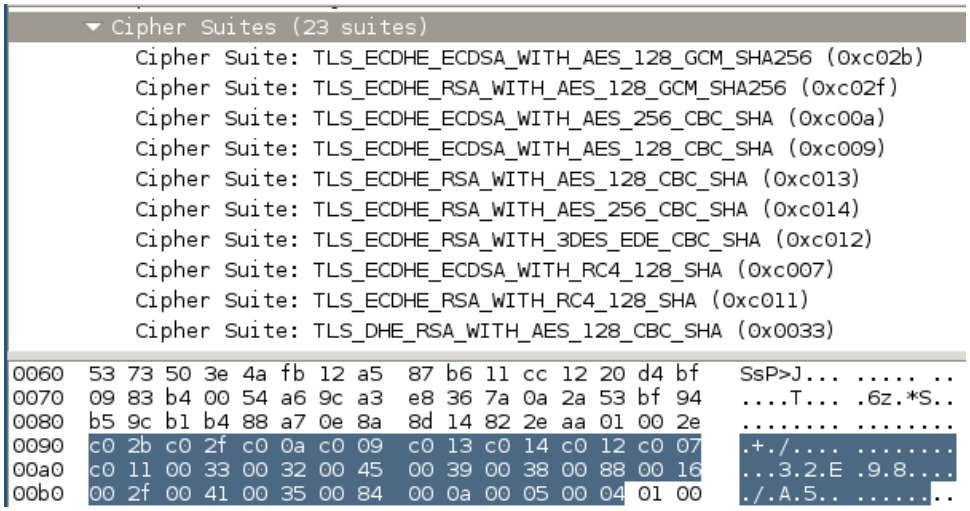
Le premier message d'une connexion SSL est le message Client Hello qui démarre le « handshake » (poignée de main). Dépliez la section « `Handshake Protocol : Client Hello` », on peut y voir un certain nombre d'information intéressantes :

- `Random` : 28 octets de données aléatoires, sera utilisé pour Diffie Hellman



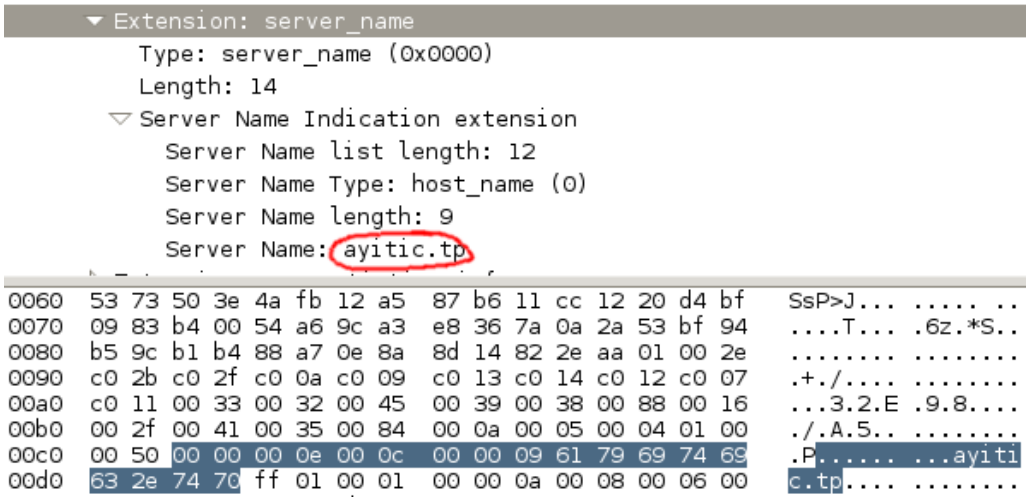
- `Session ID` : Identifiant de session pour éviter de refaire une négociation entière à la prochaine connexion

- `Cipher Suites` : C'est une liste des algorithmes de chiffrements que le navigateur supporte.



Ils sont classés par ordre de préférence on peut voir que celui que Firefox préfère est : `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256` (Voir la section terminologie pour une explication détaillées de chaque termes). Bien sur Firefox ne sait pas encore que le serveur a un certificat généré avec RSA (et ne peut donc pas utiliser ECDSA).

- `Extension: server_name` : Ce champs préciser au serveur que le navigateur cherche à accéder à la page ayitic.tp. Cela est pratique car l'échange SSL est fait avant la première requête HTTP, or un même serveur peut héberger plusieurs sites web différents (et donc plusieurs certificats). Ce champs permet au serveur d'envoyer le bon certificat SSL correspondant à la page demandé.



IV. Server Hello

Observez le message SSL suivant « Server Hello » qui est la réponse du serveur. C'est un message volumineux qui signifie que le serveur a accepté notre requête SSL. Ce message SSL contient quatre sections :

- Server Hello
- Certificate
- Server Key Exchange
- Server Hello Done

Question 17 : Observez la section Server Hello, quel Cipher Suite a été choisie par le serveur ?

Question 18 : En comparant avec le certificat sur le navigateur Firefox, que contient le champs « `Encrypted` *» dans la section «* `Certificate` *» du message* `Server Hello` *?*

Les sections Change_Cipher_Spec sont utilisés pour prévenir le receveur que les données seront chiffrés en utilisant le Cipher précédemment définis (question 17). Les données sont donc ensuite chiffrées avec le Cipher définis et envoyés dans des enregistrement de type `Application Data`.

Cette partie nous a simplement permis de demystifier un échange SSL. Elle met en évidence que les certificats ne sont utilisés que pour l'identification et l'authentification des données mais ne sont pas utilisés pour effectivement chiffrer les données. L'algorithme utilisé pour cela est négocié de part et d'autre par le navigateur et le serveur (`Cipher_Suite`). Nous allons maintenant apprendre à configurer le serveur de façon à choisir des algorithmes de chiffrement et de signature robustes.

V. Configuration Avancée de SSL

Comme nous l'avons vu précédemment, lors d'une connexion SSL à un serveur, le navigateur utilise une certaine version de SSL/TLS et soumet une liste de Cipher (algorithme de chiffrement) compatible parmis laquelle le serveur en choisira un. Nous allons configurer SSL de façon à :

- Utiliser des protocoles sûrs en rejettant SSLv1 et SSLv2 qui sont des versions obsolètes
- Utiliser des Cipher Suite sûres

Nous allons éditer la configuration de Apache de façon à désactiver SSLv2 qui n'est plus sûrs et qui est en plus obsolètes. Ouvrez le fichier de configuration SSL de Apache qui se trouve dans le chemin suivant : `/etc/apache2/mods-available/ssl.conf`. Vérifiez que les options de configuration suivantes soient corrects :

```
SSLProtocol all -SSLv2
```

Voir la page https://httpd.apache.org/docs/2.2/mod/mod_ssl.html pour plus d'information et plus particulièrement la section SSLProtocol

Cette directive va donc autoriser toutes les versions de SSL excepté SSLv2. Le navigateur pourra donc dialoguer avec le serveur à l'aide du protocole SSLv3, TLSv1.0, TLSv1.1 et TLSv1.2

VII. Utiliser des Chiffrements sûrs

Afin d'avoir des communications sûrs et robustes nous allons utiliser des algorithmes de **chiffrement** qui utilisent des clés d'au moins 128 bits. Des clés plus petites ne fournissent pas une sécurité suffisante. Nous voulons également nous assurer que les communications soient **authentiques** c'est à dire qu'on communique avec la bonne personne. C'est le rôle des certificats mais nous devons nous assurer que le certificat soit bien utilisé pour signer les données échangées. Nous devons donc éviter les protocoles suivants :

- ADH (Anonymous Diffie Hellman) : Ne fournit pas d'authentification
- NULL cipher : ne fournit pas de chiffrement
- Chiffrements faibles utilisant des clés de 40 ou 56 bits
- L'algorithme RC4 : Algorithme faible à éviter
- 3DES : ne fournit que 108 bits de sécurité qui est inférieur au minimum recommandé.
- MD5 : Cet algorithme de hachage est sensible à l'attaque de collision

Nous pouvons choisir les algorithmes de chiffrements à utiliser dans `/etc/apache2/mods-available/ssl.conf`

Vérifiez que les options de configuration suivantes soient corrects :

```
SSLCipherSuite HIGH:MEDIUM:!aNULL:!MD5
```

Voir la page https://httpd.apache.org/docs/2.2/mod/mod_ssl.html pour plus d'information et plus particulièrement la section SSLCipherSuite

Si vous administriez un vrai serveur WEB, vous pourriez tester la configuration de votre serveur sur https://www.ssllabs.com/ssltest/

Conclusion

Durant ce TP, nous avons appris à générer, signer, distribuer et vérifier des certificats tant du côté d'un internaute que du côté d'un administrateur. Nous avons également vu comment sécuriser SSL de façon à éviter l'utilisation de protocoles et d'algorithmes obsolètes. Ces connaissances peuvent être appliquées à n'importe quel service utilisant SSL/TLS. À titre d'exemple, les protocoles suivant peuvent fonctionner de façon sécurisé au dessus de SSL/TLS :

- SMTP: pour envoyer/recevoir des emails (SMTPS)
- IMAP: pour lire ses emails (IMAPS)
- POP : pour lire ses emails (POPS)
- FTP : pour transférer des fichiers de façon sécuriser (SFTP)
- LDAP : pour communiquer avec un carnet d'adresse (LDAPS)
- XMPP : protocole de communication (ex : googletalk)

Sur la clé USB qui vous a été fournis, vous trouverez de la documentation dans le répertoire « Documentation Ayitic 2014 » situé sur le Bureau de la distribution Kali Linux. Vous trouverez entre autre un document intitulé « SSL_TLS – Deployment Best Practice.pdf ». Bien qu'il soit en anglais il contient un certain nombre de conseils sur les erreurs à éviter lors de la configuration de SSL.

Terminologie

I. Certificat

D'un logiciel à un autre les termes et les formats requis pour fonctionner peuvent être différents , Voici une liste des extensions et terminologies que l'on peut rencontrer quand on commence à manipuler des certificats.

Extension	Signification
.**csr**	Certificate Signing Request. C'est un fichier contenant une requête pour signer un certificat. Ce fichier suit la norme PKCS10 qui est définit dans la RFC 2986. Un fichier **csr** inclut tout les détails du certificat à signer comme le nom de l'organisation, le pays, la clé publique etc.. L'authorité qui reçoit un **csr** peut signer le certificat et retourner un certificat qui peut lui même exister sous plusieurs formes.
.**pem**	Définit dans les RFC 1421,1422,1423,1424, **pem** désigne en réalité un format pour spécifier un conteneur qui peut contenir un simple certificat publique (c'est le cas pour les certificats utilisés par Apache /etc/ssl/certs) mais peut également contenir une chaîne de certification complète contenant la clé publique, la clé privé et le certificat root (RootCA). Dans ce format tout est encodé en **Base64** (ascii lisible avec un éditeur de texte). Le nom **pem** provient de Privacy Enhanced Email, une méthode qui a été abandonnée pour sécuriser les mails mais dont le format de conteneur a survécu.
.cert .crt .cer	Un certificat au format **pem** (plus rarement .**der** voir plus bas)
.**key**	Un fichier .**key** est un fichier au format **pem** ne contenant souvent qu'une clé privée. Ce n'est pas vraiment un nom standardisé mais vous le trouverez souvent dans les configuration apache (dans `/etc/ssl/private`). Les permissions sur ces fichiers sont très importantes (ils ne doivent surtout pas être accessible en lecture par d'autre utilisateur) et certain programme refuseront de le chargé si les permissions ne sont pas stricts.
.pkcs12 .pfx .p12	Originalement définit par RSA dans le standart Public-Key Cryptography Standars (**pkcs**), le « 12 » est une variante de Microsoft. Ces fichiers contiennent à la fois le certificat publique et privée. Contrairement aux conteneurs **pem**, ceux là sont chiffrés. Ils peuvent être convertis en **pem** avec openssl exemple : `openssl pkcs12 -in file.p12 -out converted.pem -nodes`

D'autre format que l'on retrouve	
.der	La même chose qu'un .**pem** mais en binaire. Un fichier .**pem** est juste un encodage **Base64** d'un fichier .**der.** Il est possible de convertir un .der en .pem avec openssl : `openssl x509 -inform der -in file.der -out converted.pem` Windows perçoit ces fichier comme des certificats. Par défaut, Windows exporte les certificats dans ce format mais avec une extension différente (comme .cert, .cer, .crt)
.p7b	Définit dans la RFC2315, c'est un format utilisé par Windows pour l'échange de certificat. Java les comprend nativement.
.crl	Une liste de révocation de certificat. Les CA peuvent en produire.

Pour résumer, Il y a 4 formats différents pour présenter des certificats et leurs composants :

- **PEM** : définit par les RFC, principalement utilisé par les logiciels open-source on les retrouvent avec l'extension .pem .key .cer .cert .crt
- **PKCS7** : Un standard ouvert utilisé par Java et supporté par Windows. Ce format ne contient pas de clé privée
- **PKCS12** : Un standard privée qui fournit une plus grande sécurité face au format PEM en clair. Ce format peut contenir des clés privées. Il est essentiellement utilisé par Windows et peut être facilement converti en un format PEM (voir plus haut).
- **DER** : Le format « parent » de PEM. C'est utile de se représenter ce format comme une version binaire du PEM. Pas vraiment utilisé en dehors de windows.

II. Algorithmes d'échanges de clé

Lors de la négociation du chiffrement utilisé, un certain nombre de terme existent pour spécifier l'algorithme d'échange de clé, l'algorithme de chiffrement et l'algorithme de signature.

- RSA: L'échange de clé RSA fonctionne en chiffrant une valeur aléatoire avec la clé publique du serveur. Cela nécessite que la clé publique du serveur soit une clé RSA et que le certificat du serveur n'interdise pas le chiffrement (à travers la directive Key Usage).

- DH_RSA: L'échange de la clé se faire au travers d'un mécanisme *Diffie-Hellman statique*. Le certificat du serveur doit avoir été signé par une autorité de certification qui utilise une clé RSA.

- DH_DSS: Comme DH_RSA, Excepté que l'autorité de certification utilise une clé DSA

- DHE_RSA: L'échange de clé est *ephemeral Diffie-Hellman*: Le serveur génère une clé publique *Diffie-Hellman dynamiquement* et l'envoie au client. Le serveur signe ce qu'il a envoyé. the server dynamically generates a DH public key and sends it to the client; the server also *signs* what it sends. La clé publique du serveur doit être de type RSA.

- DHE_DSS: Comme DH_RSA, Excepté que l'autorité de certification utilise une clé DSA

- DH_anon (ADH): Il n'y a pas de certificat, le serveur utilise un échange Diffie-Hellman qu'il peut générer dynamiquement. Comme il n'y a pas de certificat pour signer les données, L'échange de clé ADH est sensible à l'attaque Man-In-The-Middle (de type sslsniff).

Il est également possible d'utiliser des algorithmes d'échanges de clés qui utilisent les courbes elliptiques :

- ECDH_ECDSA: Comme DH_DSA, mais avec des courbes elliptiques. La clé publique du serveur doit être une clé ECDH dans un certificat signé par un CA lui même utilisant une clé publique ECDSA.

- ECDH_RSA: comme ECDH_ECDSA, mais l'autorité de certification CA utilise une clé RSA

- ECDHE_ECDSA: Le serveur envoie une clé Diffie Hellman générée dynamiquement et la signe avec sa propre clé qui doit être de type ECDSA. Cela est équivalent à DHE_DSA mais avec des courbes elliptiques à la fois pour la clé et pour la signature.

- ECDHE_RSA: Comme ECDHE_ECDSA, mais la clé du serveur est une clé RSA utilisé pour signé la clé Diffie-Hellman Ephémère

- ECDH_anon (AECDH) : Comme ADH mais avec des courbes elliptiques.