Travaux pratiques de cryptologie

PKI, tunnel SSL, Application à la sécurisation WEB, Apache.

UNIVERSITE DE LA ROCHELLE

26 septembre 2018

Michel Ménard / Département informatique:

Travaux pratiques de cryptologie

PKI, tunnel SSL, Application à la sécurisation WEB, Apache

La cryptologie

Objectifs

Cet atelier présente l'approche PKI. Concrètement vous apprendrez à créer des certificats et à instaurer une PKI interne. Vous apprendrez également à dépanner des liaisons SSL ou des échanges S/MIME. Enfin vous étudierez le logiciel Stunnel qui permet de créer des tunnels SSL

Contenus

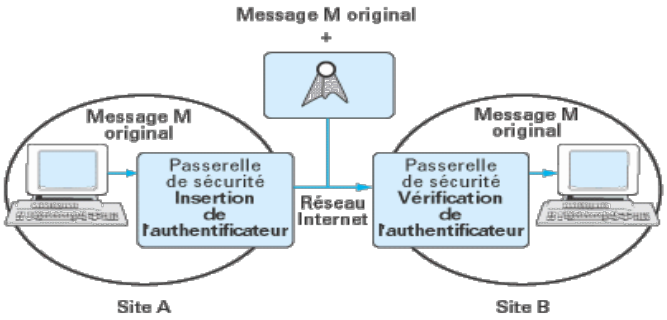
Les certificats x509

PKI

SSL

La commande Stunnel

S/MIME

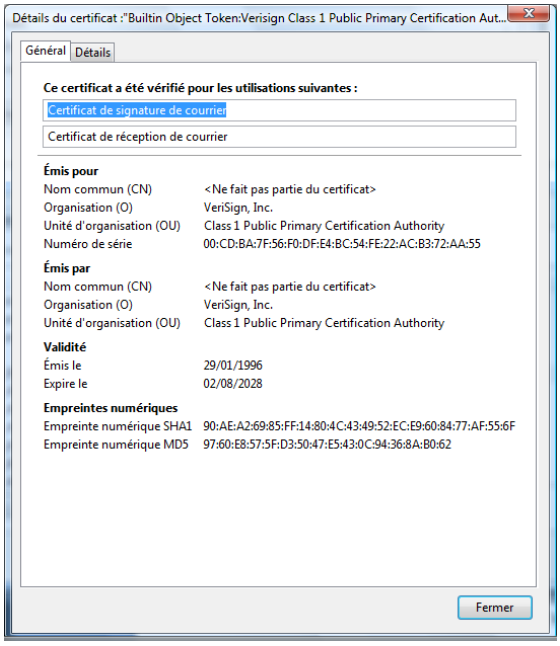


L'authentificateur est introduit par la passerelle de sécurité du site A émetteur et extrait/vérifié par l'équipement du site B récepteur.

Les certificats x509

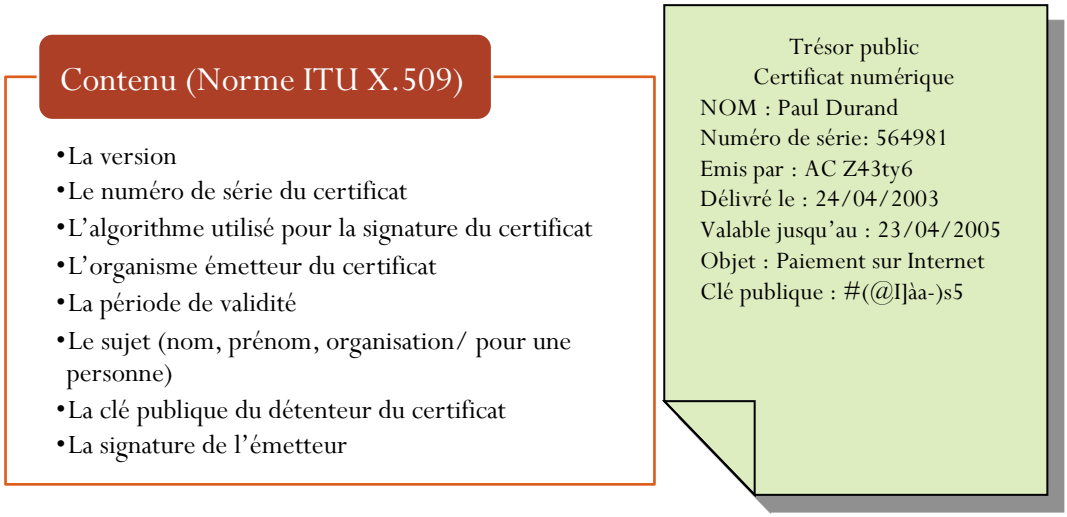
Un certificat est un document qui comprend essentiellement une clé publique et des renseignements sur le propriétaire de cette clé, le tout signé avec la clé privée d'un organisme reconnu, un CA (Certification Authoritie). Un certificat permet de dialoguer de manière sûre (chiffrée) avec la personne ou la société qu'il décrit. Si l'on possède le certificat du signataire (CA) et que l'on a confiance en lui, on a l'assurance de l'identité du propriétaire de la clé.

Le logiciel firefox contient plusieurs certificats de CA, dont celui de la société Verisign.



L'ISO a établi un protocole, baptisé x509, permettant l'authentification sur un réseau. Ce protocole utilise des certificats au format normalisé. Les certificats x509 ont été utilisés au début pour la sécurisation du courrier électronique dans le cadre du protocole PEM (Privacy Enhanced Mail), l'ancêtre de S/MIME. Maintenant ils sont utilisés aussi bien dans la sécurisation du Web (via SSL) que dans bien d'autres domaines. Les différents champs présents dans un certificat sont présentés sur la figure ci-dessous.

Contenu d'un certificat



Un certificat est un stocké dans un fichier. Les formats les plus souvent utilisés sont les suivants :

DER — Le format DER, de l'OSI, est le format binaire qui traduit l'ASN.1

PEM — Le format PEM est un format ASCII, codé en Base64. Le texte est encadré de lignes « BEGIN CERTIFICATE », « END CERTIFICATE »

PKCS#7 — Le standard RSA est principalement utilisé pour les certificats des utilisateurs. Un fichier PKCS#7 contient non seulement le certificat d'un utilisateur mais aussi sa clé privée normalement protégée par un mot de passe.

La description du sujet (possesseur de la clé publique) et du CA peuvent utiliser les abréviations suivantes :

C (Country) : pays

S (State) ou P (Province) : état ou province

L (Locality) : ville

O (organization) : société

OU (Organization Unit) : le service

CN (Common Name) : le nom du sujet ou l'adresse DNS du serveur

E (E-mail) : adresse e-mail

Un certificat d'un CA intermédiaire peut signer d'autres certificats. Mais il est lui-même signé par un CA hiérarchiquement supérieur, et ainsi de suite. Au sommet de cette pyramide, il y a les certificats des CA racines qui sont auto signés : le sujet et le signataire sont les mêmes.

Un serveur Apache par exemple doit envoyer au client non seulement son certificat, mais aussi les certificats des CA intermédiaires. Le client n'a besoin de posséder que le certificat racine.

Dans la troisième version du protocole x509v3, des extensions sont apparues. Elles permettent à un certificat de contenir des champs additionnels. Chaque champ a un nom, une valeur et un drapeau indiquant s'il est critique. Si ce drapeau est positionné et qu'une application ne sait pas interpréter le champ, elle doit rejeter le certificat.

Exemple :

Visualiser le contenu d'un certificate (au format PEM)

```
openssl x509 -in certif.pem -text -noout
```

Créer un certificate auto-signé

Créer un couple de clé publique/privée

```
openssl genrsa -out cle.pem 1024
```

Créer une requête de certificate

```
openssl req -new -key cle.pem -out cert.req
```

Signer la requête (ce qui crée le certificat)

```
openssl x509 -req -in cert.req -signkey cle.pem -out cert.crt
```

PKI

Une PKI (Public Key Infrastructure) est une organisation gérant les certificats x509 afin d'instaurer la confiance dans les échanges de données, principalement en permettant l'échange de clés publiques et l'identification des ordinateurs et des individus.

Le CA (Certification Authority) est la clé de voûte de l'édifice :

- Il crée les certificats (en les signant)
- Il doit vérifier l'authenticité des données présentes dans une requête de certificat. Ainsi il les garantie via sa signature.

Il existe deux types de CA

- Les CA publiques : leurs certificats vérifient l'identité des serveurs sur Internet
- Les CA privés : c'est un CA interne à une société. Ils permettent de créer une PKI interne.

Une CRL (Certificate Revocation List) contient au niveau d'un CA, la liste des certificats révoqués qui n'ont pas encore expiré. La publication des CRL est optionnelle. Il existe plusieurs méthodes de publication, par exemple LDAP. La révocation peut avoir lieu si la compromission d'une clé a été détectée (si en effet un pirate obtient la clé privée d'un serveur, il peut écouter toutes les transactions de celui-ci). Une limitation des CRL est le délai de

mise à jour des révocations. Pour réduire ce délai, il est possible d'utiliser en lieu et place le protocole OCSP (Online Certificate Status Protocol). Grâce à ce protocole, une application peut s'adresser à un serveur OCSP et lui demander l'état d'un certificat. Elle précise son numéro de série et le serveur répond :

- Le certificat est valide
- Le certificat est révoqué
- Ce certificat m'est inconnu

Les procédures à résoudre pour instaurer une PKI interne :

- Procédures de demandes de certificats
- La création des certificats et leur renouvellement
- L'installation des certificats racine sur les postes
- La publication des certificats, par exemple dans une base LDAP
- Les règles de sécurité associées à la protection des clés privées
- Les règles de sécurité associées à la protection du poste abritant le CA
- La gestion de la révocation des certificats

Les commandes :

openssl : cette commande possède plusieurs sous commandes permettant la mise en place d'une PKI :

x509, req, ca, verify, crl, ocsp …

Pour en savoir plus : PKI Open Source - Déploiement et administration, par Christophe Cachat, David Carella

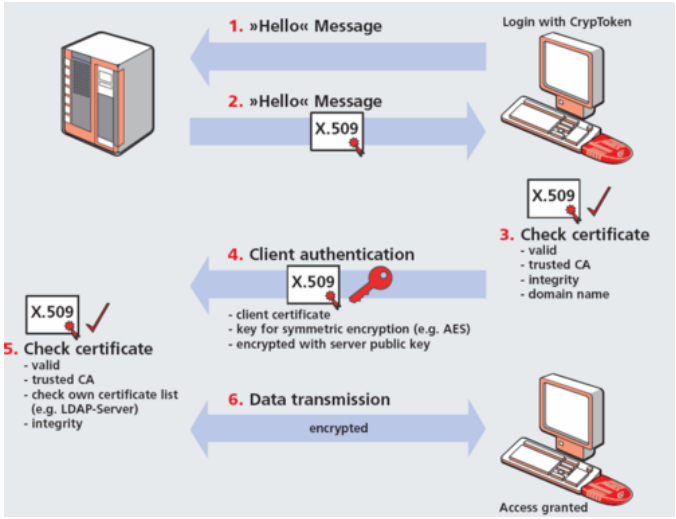
SSL

Le protocole cryptographique SSL (Secure Sockets Layers) a été créé par la société Netscape pour supporter des échanges sécurisés et authentifiés entre un navigateur Web et un serveur Web. En fait SSL peut être utilisé pour toute application reposant sur TCP, exemple notable, LDAP.

Après la version 2 du protocole, Microsoft créa PCT. La version 3 de SSL incorpore les possibilités de PCT. Cette version 3 est à la base du protocole normalisé TLS (Transport Layer Security) de l'IETF.

SSL réalise les objectifs suivants :

- Authentification du serveur
- Authentification du client (peu fréquemment utilisé)
- Confidentialité des échanges
- Contrôle de l'intégrité des échanges.



- *Aperçu du fonctionnement de l'authentification client SSL - Source : WIKIPEDIA*

Le protocole SSLv3 qui repose normalement sur TCP, est composé de deux sous-couches :

- Message (SSL message Layer)
- Enregistrement (SSL records)

La sous-couche enregistrement est la sous-couche basse : le client et le serveur échangent des enregistrements d'une taille maximale de 16383 octets de données. Les enregistrements ne sont pas liés au découpage de niveau supérieur (sous-couche message), et ainsi un enregistrement peut correspondre à plusieurs messages et inversement. Chaque enregistrement contient les informations suivantes :

- Le type de contenu
- La version du protocole
- La longueur
- Les données, cryptées et éventuellement compressées
- Le MAC (Message Authentification Code)
  Le MAC dépend d'un secret détenu par le client et le serveur, du numéro de l'enregistrement et des données. Le MAC joue donc un rôle très important du point de vue de la sécurité, car il authentifie l'enregistrement et est le garant de l'intégrité des données. Il évite aussi un certain nombre de problèmes comme la réutilisation de message (Replay).

La sous-couche message est composée de plusieurs protocoles :

- Le protocole de prise de contact (Handshake)
- Le protocole Alerte (Alert)
- Le protocole de changement de stratégie de cryptage (ChangeCipherSpec)
- Le protocole de transfert de données

Le protocole de prise de contact (Handshake)

Les objectifs de la prise de contact (Handshake) entre le client et le serveur sont les suivants :

- Négocier la version du protocole SSL
- Sélectionner les algorithmes de chiffrement
- Authentifier le serveur et/ou le client
- Echanger un secret qui permet de générer une clé de session pour ensuite poursuivre la conversation de manière chiffrée.

Pour s'authentifier, le serveur envoie au client un certificat x509. Le client l'examine, et vérifie notamment :

- Si le certificat possède une date valide
- S'il possède le certificat du CA signataire du certificat du serveur (ou le certificat du CA qui identifie le certificat du CA intermédiaire, fourni par le serveur, qui identifie le certificat du serveur, et ainsi de suite).
- S'il l'étape b aboutit, il peut vérifier la signature du certificat.
- Via une requête reverse-DNS, il vérifie si l'adresse DNS du serveur correspond bien à son adresse IP.
  Si le client est un navigateur web, il informe l'utilisateur des problèmes éventuels et lui demande s'il doit continuer la session. Il faut avoir à l'esprit que même si l'on n'a pas confiance dans un certificat, il transporte la clé publique du serveur et permet ainsi d'établir une liaison sécurisée (chiffrée).

Le déroulement du protocole

Tout échange entre le client et le serveur commence avec ce protocole.

1- Un client accède au site via une URL sécurisée, par exemple en Web : https://le-site/le-document
2- Le logiciel client envoie le message ClientHello. Il contient :
    a. La version de SSL la plus haute qu'il comprend
    b. Une valeur aléatoire
    c. Le numéro de la session (SessionID)
    d. Son paramétrage de sécurité (méthodes cryptographiques supportées, taille des clés supportées, méthodes de compression supportées…)
3- Le serveur répond en envoyant le message ServerHello. Ce message est comparable au message ClientHello.
4- Le serveur envoie ensuite son ou ses certificats x509. Par exemple, il envoie le certificat qui l'identifie et la chaîne des certificats des CA (le certificat de l'autorité intermédiaire qui a signé son certificat, le certificat qui identifie l'autorité intermédiaire et ainsi de suite). Si le serveur n'a pas de certificats, il envoie le message Echange de clés. Ce message transporte alors les informations associées au protocole Diffie-Hellman. Eventuellement, le serveur envoie un message de requête de certificat qui demande au client de s'authentifier.
5- Le client envoie son certificat si le serveur lui en a demandé un.
6- Le client envoie le message Echange de clés dont le contenu est différent en fonction de l'algorithme à clés publiques utilisé.
7- Le client envoie le message Vérification de certificat si on lui a demandé un certificat et que celui-ci ne permet pas que de réaliser des signatures.
8- Le client et le serveur échangent des messages de changement de stratégie de cryptage. Après cet échange, le reste du dialogue est chiffré.
9- Le client et le serveur échangent des messages cryptés Prise de contact finie qui permettent aux correspondants de vérifier s'ils sont synchronisés/

Le protocole de changement de stratégie de cryptage

Ce protocole est utilisé pour changer de stratégie de cryptage. Il peut intervenir à tout instant. Il correspond à une négociation entre le client et le serveur d'un algorithme de cryptage et de la clé associée.

Le protocole alerte

Le protocole alerte transporte les messages d'alertes. Ce type de message est composé de deux parties : le niveau d'alerte et la description d'alerte. Il y a deux niveaux d'alertes :

- Attention (Warning)
- Fatal. Ce message entraîne la fin de la session SSL.

Pour éviter une surcharge du serveur, une session SSL peut continuer si le numéro de session SSL (ID) envoyé par le client correspond à une session en cours. Si le numéro de session n'est pas connu, le serveur initialise alors une nouvelle session.

Les sous-commandes SSL d'OpenSSL

Les sous-commandes suivantes d'openssl permettent de suivre à l'écran les étapes du protocole SSL, et ainsi d'aider au dépannage du dialogue client/seveur SSL.

s_server	Simule un serveur Web/SSL
s_client	Simule un client Web/SSL

Exemples d'utilisation d'openssl : se connecter à un serveur Web/SSL

openssl s_client -connect www.redhat.com:443

Après cette commande, le début du protocol SSL apparaît. On peut faire une requête http, par exemple en tapant la commande suivante suivit de deux validations : GET / http/1.0

Faire office de serveur Web/SSL

Il faut spécifier le port réseau (443), fournir un certificat (cert.cer) et la clé privée (cle.pem) associée :

openssl s_server -accept 443 -cert cert.cer -key cle.pem -www -state

Options communes de s_server et s_client

-debug	affiche plus d'informations
-quiet	affiche moins d'informations
-state	affiche l'état de la session SSL
-CApath chemin	le répertoire qui contient les certificats des CA
-CAfile chemin	le fichier qui contient les certificats des CA
-verify [prof]	active la vérification des certificats. On peut préciser la profondeur du niveau de vérification
-showcerts	montre la chaîne des certificats
-cipher liste	liste les algorithmes de chiffrement supportés
-crlf	convertit un caractère LF en CR+LF

Les options de s_client

-connect host :port	l'adresse et le port du serveur, par défaut localhost :4433
-cert certif	le certificat qui doit utiliser le client si on lui en demande un
-reconnect	essaye de se reconnecter cinq fois au serveur avec le même ID de session
-showcert	affiche l'ensemble des certificats, par défaut, affiche seulement le certificat du Serveur
- prexit	affiche des informations sur la session à la fin du programme

Les options de s_serveur

-accept port	le port d'écoute du serveur, par défaut 4433
-context id	force la valeur du context ID
-cert certif	le fichier qui contient le certificat du serveur
-key fichier	le fichier qui contient la clé privée du serveur
-nocert	on n'utilise pas de certificats, on utilise le protocole Diffie-Hellman pour générer une clé de session
-www	envoie des informations au client sous la forme html et ainsi visibles à partir d'un navigateur
-WWW	émule un serveur Web, les pages sont recherchées à partir du répertoire courant.

La commande Stunnel

Le programme Stunnel est un logiciel libre sous licence GPL qui sert à établir un tunnel SSL. Celui-ci peut transporter des services TCP quelconques (SMTP, POP, IMAP, LDAP …). Le programme Stunnel joue le rôle d'un client SSL ou d'un serveur SSL selon sa configuration. Stunnel existe pour Unix et Windows.
On peut utiliser Stunnel pour construire un tunnel complet : un client Stunnel dialogue avec un serveur Stunnel. Le tunnel ainsi créé permet à un client et à un serveur (IMAP par exemple) de dialoguer de manière sécurisée. Stunnel peut être utilisé également pour créer des demi-tunnel SSL : le tunnel SSL est créé d'un coté par Stunnel et de l'autre par une application cliente ou serveur qui supporte SSL. Ainsi, par exemple un client de messagerie supportant SSL peut dialoguer avec un serveur IMAP qui lui ne supporte pas SSL par le truchement de Stunnel.

stunnel.conf : Le fichier de configuration de stunnel
stunnel.pem : le certificat et la clé privée

Exemple :
Le fichier de configuration suivant offre un support SSL au démon local imapd. Ce démon est activé en cas de demande de connexion d'un client

```
[imapd]
        accept=993
        exec=/usr/sbin/imapd
        execargs=imapd
```

L'exemple suivant est identique au precedent, à ceci près que le demon imapd est déjà actif

```
        accept=993
        connect=localhost:143
```

S/MIME

S/MIME est un protocole cryptographique normalisé par l'IETF, conçu pour sécuriser un courrier électronique au format MIME. Il assure la confidentialité du message et/ou l'authentification de l'émetteur.

S/MIME est supporté notamment par les logiciels de messagerie Outlook, Evolution et Thunderbird.

S/MIME n'est pas le seul protocole qui permette d'assurer la sécurité d'un e-mail. Il existe également les protocoles PGP et PEM.

S/MIME recommande l'usage du DES, du Triple-DES et de RC2 pour assurer le chiffrement du courrier.

Pour l'authentification, S/MIME utilise l'algorithme RSA

Pour les signatures, S/MIME utilise SHA-1 avec RSA

Les clés publiques sont mémorisées dans des certificats x509. Ces derniers sont logiquement gérés à travers une PKI.

S/MIME est donc adapté à la sécurisation du courrier du personnel d'une entreprise.

La commande openssl dispose de la sous-commande smime qui implémente le protocole S/MIME. Elle sert essentiellement à effectuer des tests et du dépannage.

La sécurisation du WEB, Apache

La sécurisation d'un serveur Web Apache passe par plusieurs lignes de défense :

- Les droits
- L'utilisation de liaisons chiffrées et authentifiées
- L'authentification des utilisateurs
- Le fait de restreindre l'accès des pages ou des applications qu'à certains postes
- L'utilisation d'applications Web sécurisées
- La minimisation, en particulier, l'installation du moins de modules possibles
- L'audit

Le serveur Apache démarre sous le compte root pour pouvoir s'associer au port 80. Ensuite il fait appel à l'appel système setiud() pour prendre les droits d'un utilisateur et d'un groupe ordinaire. Sa configuration spécifie lesquels. Si des applications sont activées par Apache, elles s'exécuteront sous ces comptes. Si les répertoires et les pages appartiennent à d'autres comptes (par exemple à root), les applications ne pourront pas modifier ou détruire les données abritées par le serveur.

Le protocole http propose deux méthodes d'authentification :

- L'authentification HTTP Basic : l'utilisateur fournit un nom et un mot de passe en clair.
- L'authentification HTTP Digest qui est de type challenge/réponse. Elle est similaire au protocole CRAM-MD5 : une chaîne aléatoire est donnée par le serveur, le client doit renvoyer une somme MD5 qui dépend de cette chaîne et de son mot de passe.

Apache dispose de plusieurs modules pour stocker la base de données d'authentification (nom et mot de passe). Certains la mémorisent dans des fichiers locaux, d'autres dans une base SQL ou un annuaire LDAP.

L'encapsulation SSL/TLS est supportée par Apache. Grâce à elle, certaines transactions sont chiffrées et éventuellement authentifiées. L'authentification porte par défaut sur l'authenticité du serveur, elle peut porter également sur celle du client.

Apache dispose d'un module de pare-feu. Il ne fait pas double emploi avec Iptables car il est configurable répertoire par répertoire, site virtuel par site virtuel, etc

Non seulement le serveur Apache doit être sécurisé mais également les applications qu'il abrite. Il est conseillé de s'adresser à des développeurs expérimentés qui ont suivit une formation particulière sur la création d'applications sûres. Une des failles majeures des applications Web est l'usage non contrôlé des données saisies dans un formulaire. Un développeur Web ne devra jamais faire confiance dans les données provenant d'un formulaire. Ces derniers pouvant contenir par exemple du code ou des instructions SQL créées par un pirate. Ces techniques sont nommées respectivement injection de code et injection SQL.

Exemple d'un module lié à la sécurité : mod_ssl (permet d'utiliser des connexions SSL/TLS. Le module est en fait une interface avec la bibliothèque OpenSSL.

Les commandes importantes :

htpasswd :	crée ou met à jour la base de données utilisée pour l'authentification Basic
htdigest :	crée ou met à jour la base de données utilisée pour l'authentification Digest

Les principales directives Apache liées à la sécurité :

Allow, Deny :	Autorise, interdit l'accès à des postes ou des réseaux. Les adresses sont sous forme DNS ou IP (mod_authz_host)
Authype :	Le mode d'autorisation (Basic ou Digest)
Require :	Restreint l'accès aux utilisateurs authentifiés ou à des utilisateurs ou à des groupes particuliers.

Les directives du module mod_ssl :

SSLCertificateFile :	Spécifie l'emplacement du certificat
SSLCertificateKeyFile :	Spécifie l'emplacement de la clé privée
SSLCipherSuite :	Spécifie les algorithmes de cryptage et/ou les tailles des clés exigées
SSLRequire :	Restreint l'accès du client par rapport à un ensemble de paramètres
SSLRequireSSL :	Oblige le client à utiliser SSL
SSLVerifyClient :	Demande le certificat x509 du client

Remarque. Les directives Apaches ont par défaut une portée globale. Si elles sont dans des directives conteneurs, elles ne s'appliquent qu'à un répertoire ou qu'à un site virtuel ou qu'à une URL donnée.

Les logiciels d'audit : nikto, Whisker, Nessus

Ateliers

Récupérer, visualiser, transcoder un certificat
Créer un certificat x509 auto-signé
Tester une liaison SSL
Créer un tunnel SSL avec Stunnel
Créer un CA, signer des certificats
Révoquer un certificat
Créer un certificat pour une personne
S/MIME
Application à la sécurisation WEB, Apache

Connectez-vous sous Linux sous votre compte utilisateur (login:tpuser, mot de passe:tpuser). Certaines fonctions nécessitent d'être root (administrateur), vous utiliserez alors la commande de terminal su - (puis mot de passe root). Vous devrez créer dans un premier temps un dossier certificat sous lequel vous travaillerez.

Manipulation 1

Créer un certificat x509 auto-signé

1. Créer un couple de clés publique/privée

   openssl genrsa -out server.key 1024

   chmod 440 server.key

2. Créer une requête de certificat
   Lorsqu'on crée la requête, un certain nombre d'informations vous sont demandées. La plus importante est le Common Name. Il faut saisir le FQDN du serveur pour lequel est destiné le certificat (localhost, ou l'adresse IP - choisissez de préférence l'adresse IP où sera lancé votre serveur).

   openssl req -new -key server.key -out server.req

   :FR ; :PC ; :La Rochelle ; :ULR ; :INFO ; : Mettre l'ADRESSE IP de l'hôte ;

   head -3 server.req

3. Créer le certificat
   Le certificat est normalement créé par un CA quand il signe la requête avec sa clé privée. Dans le cas d'un certificat auto-signé, c'est le même organisme qui crée la requête qui la signe.

   openssl x509 -req -days 365 -in server.req -signkey server.key -out server.crt

4. Afficher le certificat et copier ensuite le fichier server.crt dans /var/www/html

   openssl x509 -in server.crt -text |head

Manipulation 2

Récupérer, visualiser, transcoder un certificat

1. Saisir le script retreive-cert.sh. **Attention** au copier-coller à partir du document pdf, certains caractères ne passent pas, comme les quotes et les - . **Vous pouvez sinon le télécharger à partir de Moodle**.

```
vi retrieve-cert.sh

# !/bin/sh

# usage : retrieve-cert.sh remote.host.name [port]

REMHOST=$1

REMPORT=${2:-443}

echo |\

openssl s_client -connect ${REMHOST}:${REMPORT} 2>&1 |\

sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p'
```

2. Récupérer un certificat et l'afficher (utiliser le site apps.univ-lr.fr).

```
sh retrieve-cert.sh apps.univ-lr.fr > univ.pem

file univ.pem

cat univ.pem
```

Pour comprendre le script, lancez dans le terminal la commande :

```
openssl s_client -connect apps.univ-lr.fr:443 | less
```

3. Visualiser un certificat qui est au format PEM.

```
openssl x509 -in univ.pem -text -noout | more
```

4. Convertir un certificat d'un format en un autre (ici de PEM à DER).

```
openssl x509 -inform PEM -in univ.pem -outform DER -out univ.der
```

5. Visualiser un certificat qui est au format DER.

```
openssl x509 -inform DER -in univ.der -text | head
```

Manipulation 3

Tester une liaison SSL

1. Activer le serveur de test.
   On donne en argument le port d'écoute (ici 5000) ainsi que le chemin du certificat et de la clé privée.
   `openssl s_server -accept 5000 -cert server.crt -key server.key -www -state`

   Visualisez les informations après une connexion en local avec un navigateur. Remarque : vous pouvez mettre fin au serveur en appuyant sur Crtl-C. (https://localhost:5000 en local)

   Que signifie le message d'avertissement ?

Visualisez le certificat à travers le navigateur. Retrouvez les informations saisies lors de la création du certificat (Manipulation 1).

Tester l'accès à partir d'un client (installez lynx via la commande apt-get install lynx) :

lynx 'https://localhost:5000' ou bien avec wget comme précisé ci-dessous :

wget --no-proxy 'https://localhost:5000'

wget --no-proxy --no-check-certificate 'https://localhost:5000'

Analyser et discuter des informations affichées (vérifiez avec votre adresse IP à la place de localhost).

3. Afficher la trace d'une session SSL avec le client de test.
   openssl s_client -connect localhost:5000

   Taper la commande suivante :

   GET / HTTP/1.0

4. Accéder à un site SSL présent sur Internet avec le programme de test

   openssl s_client -connect apps.univ-lr.fr:443

   (vous devez avoir le message Verify return code 0 ok). Taper la commande GET / HTTP/1.0 (et deux fois entrée)

5. Utilisation d'un protocole obsolète (Diffie-Hellman).
   Activer un serveur SSL n'utilisant que le protocole Diffie-Hellman (Pensez à arrêter le serveur précédent)
   openssl s_server -accept 5000 -nocert -www -state

   Essayer de se connecter à partir d'un client

   wget --no-proxy 'https://localhost:5000' (ou avec un navigateur)

   Analyser les messages apparaissant sur l'écran de l'ordinateur

Manipulation 4

Créer un tunnel SSL avec stunnel

1. Le serveur stunnel doit être installé. Sinon, installez-le : sudo apt-get install stunnel

   Idem pour le serveur telnetd : sudo apt-get install telnetd

   Pour vérifier si le serveur telnetd est bien installé : netstat –an | grep ':23'. Il doit être à l'écoute.

2. Placez-vous dans le dossier où se trouvent les certificats générés précédemment et copiez les (certificat et clé privée du serveur) dans le répertoire de configuration de stunnel
   sudo cp ./server.{crt, key} /etc/stunnel

3. Configurer
   sudo vi /etc/stunnel/stunnel.conf

   cert=/etc/stunnel/server.crt

   key=/etc/stunnel/server.key

pid=/var/run/stunnel.pid

debug=7

output=/var/log/stunnel.log

[s1]

accept=5000

connect=localhost:23

et activer le serveur :

sudo stunnel

tail /var/log/stunnel.log

netstat -an | grep -e ':5000' -e ':23'

Remarque

On peut donner en argument de la commande le chemin du fichier de configuration

stunnel /etc/stunnel/stunnel.conf

4. Configurer un client (ouvrir une deuxième fenêtre de contrôle sur localhost)
vi /etc/stunnel/cli.conf

pid=/tmp/stunnel.pid

debug=7

output=/tmp/stunnel.log

client=yes

[s2]

accept=localhost:4000

connect=localhost:5000 (ou bien mettre le numéro de la machine où se trouve le serveur)

5. Créer un couple de clés publique/privée.
sudo stunnel /etc/stunnel/cli.conf

netstat -an | grep ':4000'

6. On utilise le tunnel SSL pour véhiculer une session telnet.

telnet localhost 4000

tpuser

~~tpuser~~

exit

cat /tmp/stunnel.log

7. Sur l'ensemble des postes, mettre fin aux processus stunnel.
   sudo pkill stunnel
8. Interpréter et représenter graphiquement la connexion réalisée. Quel est son intérêt ? Aidez-vous de wireshark.

Manipulation 5

Créer un CA, signer des certificats

1. Se placer dans /root. On va créer l'environnement du CA (connectez-vous avec sudo –i)

   mkdir minica

   cd minica

   mkdir certs private

   chmod g-rwx,o-rwx private/

   echo "01" > serial

   touch index.txt

2. Créer la configuration paramétrant le futur certificat racine : vi create_root.cnf
   Le fichier se trouve également sous moodle

```
[ req ]
  default_bits            =2048
  default_keyfile         =/root/minica/private/cakey.pem
  default_md              =sha256

  prompt                  =no
  distinguished_name      =root_ca_dn
  x509_extensions         =root_ca_ext

[ root_ca_dn ]
commonName                =CAulr
stateOrProvinceName       =PC
countryName               =FR
emailAddress              =minica@univ-lr.fr
organizationName          =ULR

[ root_ca_ext ]
basicConstraints          =CA:true
```

3. Créer le certificat racine.
   export OPENSSL_CONF=/root/minica/create_root.cnf

   openssl req -x509 -days 365 -newkey rsa:2048 -out cacert.pem -keyout cakey.pem -outform PEM

   ~~secret ; secret~~

4. Afficher le certificat.
openssl x509 -in cacert.pem -text -noout |more

unset OPENSSL_CONF

copier le fichier (clé privée) cakey.pem dans le dossier /root/minica/private/ (voir le fichier de configuration : create_root.cnf)
copier le fichier (certificat) cacert.pem dans le dossier /root/minica/ (voir le fichier de configuration openssl.cnf ci-après). Il se peut qu'il existe déjà. C'est juste pour s'en assurer !

5. Le CA signe des certificats. Un client crée une requête de certificat et envoie sa requête au CA.
openssl genrsa -out cle.pem 1024

openssl req -new -key cle.pem -out linux1.req : FR ; PC ; La Rochelle ; ULR ; INFO ; num_ip ;

Le CA crée un fichier de configuration pour les clients

vi /root/minica/openssl.cnf (Il existe également sous Moodle !)

```
[ ca ]
  default_ca=exemple_ca
[ exemple_ca ]
 dir                    =/root/minica
 certificate            = $dir/cacert.pem
 database               = $dir/index.txt
new_certs_dir           = $dir/certs
private_key             = $dir/private/cakey.pem
serial                  =$dir/serial

default_crl_days        =7
default_days            =365
default_md              =sha256

policy                  =ca_policy
x509_extensions         =certificate_extensions

[ ca_policy ]
commonName              =supplied
stateOrProvinceName     =supplied
countryName             =supplied
emailAddress            = optional
organizationName        = supplied
organizationalUnitName  =supplied

[ certificate_extensions ]
basicConstraints        =CA:false
```

export OPENSSL_CONF=/root/minica/openssl.cnf

c. Le CA signe la requête (vous devez vous trouver dans le dossier où se trouve linux1.req)

openssl ca -in linux1.req

~~secret,~~ y, y

more index.txt

more serial

d.Vérifier un certificat – **Attention , placez-vous toujours dans minica**.

D'abord le visualiser : openssl x509 -in certs/01.pem -text -noout | more

Il s'agit bien du certificat pour le client linux1 (et donc associé à la requête précédente linux1.req)

openssl verify -CAfile cacert.pem certs/01.pem

faire attention, la ligne précédente dépend où vous vous trouvez (ex : si vous êtes actuellement dans le dossier minica, vous devez effectivement taper : openssl verify -CAfile cacert.pem certs/01.pem

6. Le CA publie son certificat
   cp cacert.pem /tmp

Manipulation 6 (NE PAS FAIRE !)

Révoquer un certificat

1. Révoquer le certificat.
   env | grep OPENSSL

   cp minica/certs/01.pem testcert.pem

   openssl ca -revoke testcert.pem

   ~~secret~~

   more minica/index.txt

2. Créer une CRL.
   openssl ca -gencrl -out exampleca.pem

   ~~secret~~

3. Visualiser une CRL.
   openssl crl -in exampleca.pem -text -noout | more

4. Convertir la CRL dans un format lisible par les navigateurs.
   openssl crl -in exampleca.pem -outform DER -out exampleca.crl

5. Mettre à jour les CRL d'un navigateur.
   a) Télécharge une CRL
   http://igc-crl.agriculture.gouv.fr/crl/Agriculture-AC-Serveurs.crl

   b) Gère la CRL
   Dans Firefox : [Edition]->[Préférences…]->[Avancé]->[Liste de révocation]

Manipulation 7

Préambule : créez deux utilisateurs Alice et guest (commande adduser).

Créer un certificat pour une personne

1. L'utilisateur crée un couple de clé privée/clé publique.
   su - guest

   openssl genrsa -out guest.key 1024

2. L'utilisateur crée une requête de certificat et la transmit au responsable sécurité
   openssl req -new -key guest.key -out guest.req

   FR ; PC ; La Rochelle ; ULR ; INFO ; Paul Guest ; localhost@univ-lr.fr

3. cp guest.req /tmp

   exit

   Le CA signe la requête (vérifiez que vous avez toujours le bon chemin : echo $OPENSSL_CONF qui doit donner /root/minica/openssl.cnf,

   sinon, vous devez précisez le fichier de configuration correct : openssl ca -config openssl.cnf -in /tmp/guest.req)

   openssl ca -in /tmp/guest.req

   ~~secret ;~~ y ; y

   cp minica/certs/02.pem /tmp/guest.crt

4. L'utilisateur construit le certificat complet (certificat + sa clé privée cryptée : format utilisé par apple par exemple pour la programmation sur ios).
   su - guest

   openssl pkcs12 -export -in /tmp/guest.crt -inkey guest.key -out guest.pfx

   ~~secret ;secret~~

5. Enfin, il le convertit au format PEM.

   openssl pkcs12 -in guest.pfx -out guest.pem

   ~~secret; password; password~~

6. Visualiser le certificat
   more guest.pem

7. L'utilisateur publie son certificat
   cp guest.pem /tmp

   exit

   Remarque

   Habituellement les certificats des utilisateurs sont disponibles dans une base de données accessible à l 'ensemble des utilisateurs. Typiquement, ils sont stockés dans une base LDAP

   Vous devez avoir dans /tmp : guest.crt, guest.pem, guest.req

Manipulation 8

S/MIME

1. Crypter un message (un e-mail).
   a. Un utilisateur crée un message. Il utilise le certificat du destinataire (il contient sa clé publique)

   su - alice

   vi msg.txt

   openssl smime -encrypt -in msg.txt -des3 -out /tmp/msg.enc /tmp/guest.crt

   exit

   more /tmp/msg.enc

   b. Le destinataire lit le message. Il doit donner sa « pass phrase » pour accéder à sa clé privée lui permettant de décoder le message.

   su - guest

   openssl smime -decrypt -in /tmp/msg.enc -recip guest.pem

   ~~password~~

2. Signer un message (un e-mail).

   a. L'émetteur guest signe un message
   vi msg2.txt // écrire un texte

   openssl smime -sign -in msg2.txt -signer guest.pem -out /tmp/msg2.sgn

   (si on avait utilisé l'option -nocerts, le certificat du signataire ne serait pas inclus)

   ~~password~~

   exit

   b. Le destinataire vérifie la signature (avec le certificat du CA) Attention cacert.pem qui se trouve dans minica doit être copié dans /tmp . Ici on vérifie également le certificat du signataire (par rapport au CA).

   su - alice

   openssl smime -verify -in /tmp/msg2.sgn -CAfile /tmp/cacert.pem

3. Extraire le certificat d'un message signé.
   openssl smime -pk7out -in /tmp/msg2.sgn | openssl pkcs7 -print_certs > emetteur.pem

   openssl x509 -in emetteur.pem

   exit

Manipulation 9

La sécurisation du Web, Apache

Préambule et préparation.

Nous allons dans cet exercice travailler avec docker. Dans un terminal, lancez toujours la commande docker avec les privilèges sudo.

Téléchargez le fichier securel3.tar : il s'agit d'une image docker préalablement configurée. La distribution est Fedora 24. Une fois téléchargée, vous pouvez, pour extraire l'image, lancer la commande :

docker load < securitel3.tar

Puis pour créer un conteneur :

docker run -it --name secu -p8080:80 -p443:443 -v $PWD:/tmp michel17:oko /bin/bash

Vous être à présent dans un terminal du conteneur : [root@XXXXXXXXXX /]

Dans un autre terminal, effectuez un sudo *docker images* et un sudo *docker ps --all* pour lister les images et les conteneurs lancés.

Dans le terminal du conteneur, lancez httpd et /usr/sbin/sshd pour lancer le serveur apache et le serveur sshd. Des warnings apparaissent. Vérifiez cependant leur lancement en utilisant :

- **netstat -an | grep ':80' (httpd lancé)**
- **netstat –an | grep ':22' (sshd lancé)**

Modifiez le fichier /etc/resolv.conf : il doit contenir sur la première ligne nameserver 10.2.40.230

Pour la suite de l'exercice, vous travaillerez soit à partir du terminal du conteneur, soit en vous connectant en ssh sur ce conteneur à l'adresse ip de celui-ci (en tant que root, mot de passe: jetson).

REMARQUE IMPORTANTE

Docker sous ubuntu ne peut actuellement lancer le service httpd, ni le service ssh. Pour la suite, le plus simple est de remplacer les commandes de lancement ou d'arrêt des services (systemctl start httpd.service, systemctl stop httpd.service, …) par httpd (lancement) et pkill httpd (arrêt) ou encore /usr/sbin/sshd (lancement) et pkill sshd (arrêt).

1. Lancement de httpd.
   httpd
   Puis : netstat -an | grep ':80'
   Et lancez ensuite un navigateur sur la machine hôte : http://localhost:8080: vous devrez avoir la page de Fedora.

2. Tester l'accès au serveur via lynx ou wget (sur le conteneur)
   Via lynx (s'il est présent) : lynx -dump 'http://localhost' | head -2
   ou wget --no-proxy http://localhost/testfile (avec testfile créé sous l'utilisateur root par touch /var/www/html/testfile).
   Via un navigateur (à partir de l'hôte : n'oubliez pas de passer par le port 8080 pour être transféré sur le port 80).

3. Visualiser les journaux (l'emplacement du fichier peut dépendre de votre configuration)
   tail /var/log/httpd/error_log
   tail /var/log/httpd/access_log

4. Modifier la configuration. Relancer le serveur
   Mettre en place une authentification pour les pages web présentes dans le répertoire club_prive. Les accès sont interdits sauf à partir du poste local.
   vi /etc/httpd/conf.modules.d/clubprive.conf

```
<Directory "/var/www/html/club_prive">

        AuthType Basic

        AuthName "Club prive"

        AuthUserFile /etc/httpd/conf/users

        Order deny,allow

        Deny from all

        Allow from 127.0.0.1 localhost et  l'adresse IP de  votre
machine liée à docker0

        Require valid-user

</Directory>
```

   réactivez le service avec la commande pkill httpd, puis httpd

5. Créer les comptes des utilisateurs autorisés (binôme)
   Utilisez vos logins et mots de passe :
   htpasswd -c /etc/httpd/conf/users mmenard (l'option -c crée le fichier, pour le second omettez-le.
   visualisez vos autorisations :
   cat /etc/httpd/conf/users

6. Créer le site Web
   mkdir /var/www/html/club_prive
   echo "<h1>Club prive</h1>" > /var/www/html/club_prive/index.html

7. Accéder au site
   Si l'on accède au site à partir du poste du binôme ou du poste local, le navigateur demande à l'utilisateur de s'authentifier. Le couple login, mot de passe permet d'accéder à la page Web. A partir d'un autre poste, l'accès est interdit (si ce n'est déjà fait, ouvrez le port 80 (http) du firewall local) .
   Si lynx est installé, on peut l'utiliser de deux manières :
   a. De manière interactive
      lynx "http://localhost/club_prive"
      Et authentifiez-vous.
   b. En mode scriptable
      lynx -dump 'http://localhost/club_prive' | head -2
      lynx -dump -auth=login:mot_de_passe 'http://localhost/club_prive' | head -2

Vous pouvez également utilisé (sur le poste du binôme ou en local): wget --http-user=Nom --http-password=mot de passe –no-proxy 'http://localhost/club_prive'

Avec le navigateur de votre choix.
Accéder au site à partir d'un poste non autorisé

Manipulation 10

Sécuriser Apache - Utilisation de SSL

1. Créer un certificat autosigné. Visualisez-le
   openssl genrsa -out server.key 1024
   chmod 440 server.key
   openssl req -new -key server.key -out server.req
   Entrer l'adresse IP du serveur sur lequel vous avez lancé httpd

   openssl x509 -req -days 365 -in server.req -signkey server.key -out server.crt

   openssl x509 -in server.crt -text -noout | more

2. Vérifier la présence du module SSL (installez le si nécessaire par yum install mod_ssl)
   rpm -q mod_ssl
3. Afficher la configuration par défaut du module SSL
   grep "^[a-zA-Z]" /etc/httpd/conf.d/ssl.conf

   (chercher localhost.crt et localhost.key ; Il s'agit des données précédentes normalement.
   Sauvegarder les fichiers localhost.crt et localhost.key.
4. Installer le certificat et la clé privée associée au bon emplacement
   cp server.crt /etc/pki/tls/certs/localhost.crt
   cp server.key /etc/pki/tls/private/localhost.key
5. Relancer le serveur et vérifier l'ouverture du port SSL

   pkill httpd puis httpd

   netstat -an | grep ':443'

6. A partir de votre poste local, tester l'accès à la page d'accueil avec lynx et avec un navigateur
   Tester d'abord en mode normal (http) puis en mode SSL (https)
   lynx -dump -auth=login:password 'http://num_ip_serveur/club_prive' | head -2 (ou bien-sûr localhost) : OK
   lynx -dump -auth=login:password 'https://num_ip_serveur/club_prive' | head -2 (ou bien-sûr localhost) : NO
   lynx -auth=login:password 'http://num_ip_serveur/club_prive': message d'avertissement mais accès OK
   lynx -auth=login:password 'https://num_ip_serveur/club_prive' : NO

   Avec un navigateur : http : OK
   Avec un navigateur : https : présentation d'un avertissement, certificat auto-signé. Téléchargez le certificat et consultez le.

7. Modifier la configuration dans clubprive.conf pour que l'accès puisse se faire également à partir d'un autre PC (celui à coté de vous). Relancer les commandes de la question 6, après avoir relancer le serveur.

   Avant l'installation

   a. (Si lynx installé, sinon utiliser un navigateur) Télécharger en mode scriptable la page SSL. L'opération doit échouer.
      lynx -dump 'https://nom_serveur/club_prive'

   Installation

   b. Récupérer le certificat du CA depuis le poste distant. Paramétrer lynx pour l'utiliser.
      scp num_ip_serveur:/root/serveur.crt /root
      export SSL_CERT_FILE=/root/server.crt
      (Vérifier bien les noms des dossiers. Vous pouvez utilisez /tmp).

   Essayer de nouveau d'attendre la page en SSL (avec lynx ou un navigateur)

8. Modifier la configuration du serveur où a été lancé Apache. Le forcer à l'utilisation de SSL
   vi /etc/httpd/conf.d/clubprive.conf

```
<Directory "/var/www/html/club_prive">

        AuthType Basic

        AuthName "Club prive"

        AuthUserFile /etc/httpd/conf/users

        Order deny,allow

        Deny from all

        Allow from 127.0.0.1 localhost et  l'adresse IP de  votre machine distante à partir de laquelle
vous  cherchez à vous connecter

        SSLRequireSSL

        Require valid-user

</Directory>
```

   pkill httpd puis httpd

   Tester à partir du poste distant (avec un navigateur ou wget)
   wget --no-check-certificate --http-user=login --http-password=password --no-proxy
   'https://num_ip_serveur/club_prive'
   L'accès doit réussir
   wget --no-check-certificate --http-user=login --http-password=password --no-proxy
   'http://num_ip_serveur/club_prive'
   L'accès doit échouer

9. Idem, mais on veut un chiffrement fort.

vi /etc/httpd/conf.d/clubprive.conf

```
<Directory "/var/www/html/club_prive">

        AuthType Basic

        AuthName "Club prive"

        AuthUserFile /etc/httpd/conf/users

        Order deny,allow

        Deny from all

        Allow from 127.0.0.1 localhost et  l'adresse IP de  votre machine distance à partir de laquelle
vous   cherchez à vous connecter

        SSLRequireSSL

        SSLCipherSuite ALL :!ADH :!LOW :!EXP :+HIGH :+MEDIUM

        Require valid-user

</Directory>
```

Vérifier la configuration et relancer le serveur

Les chiffrements LOW (DES simple), ADH (Anonymous Diffie Hellman) ou EXP (40 bits) sont interdits. Sont autorisés tous les chiffrements (ALL), y compris HIGH (triple DES) et MEDIUM (chiffrement 128 bits).

Tester. L'accès lynx doit échouer. Les chiffrements proposés par Lynx ne conviennent pas à Apache. Par contre l'accès par Firefox fonctionne.

10. Visualiser (sous le poste serveur) les journaux d'Apache concernant SSL
    tail -1 /var/log/httpd/ssl_error_log
    tail -1 /var/log/httpd/ssl_request_log
    tail -1 /var/log/httpd/ssl_access_log

    supprimer dans le fichier de configuration la ligne :

    SSLCipherSuite ALL :!ADH :!LOW :!EXP :+HIGH :+MEDIUM

Manipulation 11

Sécuriser un serveur : renforcer la sécurisation de l'application

1. Supprimer les bannières
   a. Afficher la bannière d'Apache à partir du poste distant
      echo -e "GET /toto.html http/1.0\r\n\r\n" | nc num_ip_pc2 80
   b. Modifier la configuration d'Apache

      cp /etc/httpd/conf/httpd.conf /etc/httpd/conf/httpd.conf.000

      vi /etc/httpd/conf/httpd.conf

      ```
      #ServerTokens OS
      ServerTokens ProductOnly
      …
      ```

```
#ServerSignature On
ServerSignature Off
```

service httpd restart

c. Afficher de nouveau la bannière à partir du poste distant. Elle n'apparait plus.
echo -e "GET /toto.html HTTP/1 .0\r\n\r\n" | nc num_ip_pc2 80

d. Afficher la bannière PHP à partir du poste distant
(remarque : si php n'est pas installé, faire yum install php php-common)
echo -e "GET /index.php HTTP/1 .0\r\n\r\n" | nc num_ip_pc2 80

e. Supprimer la bannière PHP
cp /etc/php.ini /etc/php.ini.000
vi /etc/php.ini
…
; expose_php=On
expose_php=Off

service httpd restart

f. Accéder de nouveau à la page PHP, la bannière n'apparaît plus.
echo -e "GET /index.php HTTP/1 .0\r\n\r\n" | nc num_ip_pc2 80

Manipulation 12

Supprimer les messages d''avertissement dû à un certificat auto-signé.

Lorsque vous vous connectez à partir d'un poste distant sur le serveur web, vous rencontrez un message d'avertissement précisant que le certificat du serveur est auto-signé, et qu'il n'est pas conseillé de vous y connecter. Proposez une solution pour supprimer cet avertissement. Réalisez-là. Dans quelle mesure cette solution ne pose pas de problème de sécurité ?