Travaux pratiques, séance 3

OpenSSL et Stunnel

SSL (ou TLS) spécifie des outils cryptographiques et des formats de représentation des objets correspondants (clés, signatures, certificats, ...). OpenSSL est un implémentation de ces spécifications. En plus d'une librairie C, on a une commande `openssl` qui permet de tester à la main quelques fonctionnalités de SSL.

Stunnel repose sur SSL et permet de créer des tunnels chiffrés, intègres et dont les extrémités sont identifiées. Ces tunnels peuvent être empruntés par (presque) n'importe quelle application.

1. Installation

OpenSSL

A priori, c'est installé sur Mandrake.

Stunnel

Voici la méthode que j'utilise (mais moi, j'ai les fichiers "include" de SSL) :

```
# tar -xzf stunnel-3.22.tar.gz
# cd stunnel-3.22
# ./configure --enable-dh
# make
# make install
```

Voilà la méthode que vous utiliserez probablement (à moins d'avoir installé le RPM "OpenSSL-devel" ?) :

```
# rpm -i stunnel-3.22-2mdk.i586.rpm
# openssl req -new -x509 -days 365 -nodes \
  -config /usr/share/doc/stunnel-3.22/stunnel.cnf \
  -out /etc/ssl/stunnel/stunnel.pem -keyout /etc/ssl/stunnel/stunnel.pem
```

Dans les deux cas, vous obtenez à la fin de l'installation un fichier `stunnel.pem` qui contient une clé privée RSA, un nombre premier pour des échanges Diffie-Hellman et un certificat d'auto-signature.

2. Utilisation basique de OpenSSL

Un peu de hachage

```
$ openssl dgst /etc/passwd                        % Haché MD5 du fichier /etc/passwd
$ echo -n "aaa" | openssl dgst -sha1              % Haché SHA1 de la chaîne "aaa"
$ openssl passwd -1 -salt Q/4YuT.T 'mypassword'   % Haché avec sel (en base-64) et MD5
```

Dans le dernier exemple, vous pouvez essayer avec le sel trouvé dans le fichier `/etc/shadow` (deuxième champ délimité par `$`) et votre mot de passe (entre `' '`). Cet algorithme de hachage est strictement identique à celui utilisé dans `/etc/shadow`.

Base-64, chiffrement symétrique et aléa

```
$ openssl enc -a < /etc/passwd > passwd64       % Conversion en base-64
$ openssl enc -d -a < passwd64                  % Retour au fichier original

$ openssl enc -e -des-cbc -k toto < /etc/passwd > passwd.des  % Chiffrement DES/CBC
$ openssl enc -d -des-cbc -k toto < passwd.des  % Déchiffrement

$ openssl rand -base64 128                      % Génère 128 octets aléatoires
```

Chiffrement RSA

```
$ openssl genrsa 1024 -out skey.pem           % Génère une clé RSA 1024 bits (privée)
$ openssl genrsa -des -out skey1.pem 1024     % Idem + chiffre la clé avec une passphrase
$ openssl rsa -modulus -noout < skey.pem      % Affiche le module de cette clé
$ openssl rsa -pubout < skey.pem > pkey.pem   % Extrait la clé publique correspondante

                                              % Chiffrement RSA/pkcs#1 :
$ echo baratin | openssl rsautl -encrypt -pubin -inkey pkey.pem > baratin.rsa
$ openssl rsautl -decrypt -inkey skey.pem < baratin.rsa     % Déchiffrement
```

Signature DSA

```
$ openssl dsaparam -genkey 1024 > dsa.pem       % Génère une clé de 1024 bits
$ openssl dsaparam -text < dsa.pem              % Montre les données de la clé
$ openssl dsa -pubout < dsa.pem > pdsa.pem      % Extrait la clé publique

$ openssl dgst -dss1 -sign dsa.pem < /etc/passwd > passwd.sign           % Signe
$ openssl dgst -dss1 -verify pdsa.pem -signature passwd.sign < /etc/passwd % Vérifie
```

Diffie-Hellman

```
$ openssl dhparam 512 > dh.pem                % Génère un corps premier, de générateur 2
$ openssl dhparam -text < dh.pem              % Montre les paramètres en clair
```

3. Certificats

Un certificat OpenSSL est une association entre une clé publique et un *Distinguished Name* (DN), c'est-à-dire des informations décrivant à qui appartient cette clé publique et à quels usages elle est dédiée. Cette association est réalisée par une signature cryptographique, à l'aide d'une clé appartenant à une autorité de certification. Un fichier représentant le certificat contiendra donc la clé publique, le DN associé, le DN de l'autorité certifiante, et la signature de cette autorité et d'autres données comme les dates de création et d'expiration du certificat.

La signature de l'autorité est censée prouver que le certificat est authentique. Mais pour vérifier cette preuve, il faut aussi posséder le certificat de l'autorité est être sûr qu'il est lui-même authentique. La vérification complète de l'authenticité d'un certificat fait donc intervenir une chaîne de certification à la fin de laquelle se trouve ce certificat et au début de laquelle se trouve un certificat racine (auto-signé) dont l'authenticité doit être affirmée par un autre moyen (inclusion directe dans le logiciel de vérification, par exemple).

On peut fabriquer et gérer des certificats avec OpenSSL (à petite échelle). Pour le DESS, j'ai créé une autorité racine et quelques certificats.

Il faut d'abord préparer les répertoires où seront les certificats créés et la clé secrète de l'autorité racine. Il faut aussi initialiser un compteur de certificats car chaque certificat généré par l'autorité doit avoir un numéro de série distinct. Puis il faut créer un fichier vide qui contiendra ensuite une liste des certificats générés.

```
# mkdir /etc/ssl/dess-si-ca      % Ici seront placées toutes les données
# cd /etc/ssl/dess-si-ca         % concernant l'autorité du DESS-SI
# mkdir certs private            % Pour les certificats et la clé secrète
# chmod go-rwx private           % La clé secrète doit le rester...
# echo '01' > serial             % Initialiser le compteur de certificats
# touch index.txt                % Initialiser la liste de certificats créés
```

Ensuite, pour avoir un nombre d'options non prohibitif à passer aux commandes OpenSSL, il faut préparer un fichier de configuration, sur le modèle de `openssl.conf`. Je l'ai appelé `/etc/ssl/dess-si-ca/dess-si.cnf`.

Fichier de configuration

```
[ ca ]
default_ca      = dess-si-ca

####################################################################

[ dess-si-ca ]

dir             = /etc/ssl/dess-si-ca     # Where everything is kept
certificate     = $dir/cacert.pem         # The CA certificate
private_key     = $dir/private/cakey.pem  # The private key

new_certs_dir   = $dir/certs              # default place for new certs.
serial          = $dir/serial             # The current serial number
database        = $dir/index.txt          # database index file.

default_days    = 365                     # how long to certify for
default_md      = md5                     # which md to use.
default_crl_days= 30                      # how long before next CRL

policy          = dess-si-ca-policy       # defined below
x509_extensions = dess-si-ca-ext          # also defined below

[ dess-si-ca-policy ]
countryName             = match           # 6 fields will appear in
stateOrProvinceName     = match           # the DN of each certificate.
localityName            = match           # The first 5 will exactly
organizationName        = match           # match the DN of ca.
organizationalUnitName  = match
```

```
commonName              = supplied        # This one varies but exists.
emailAddress            = optional        # Only if you want it.

[ dess-si-ca-ext ]
basicConstraints = CA:true   # Root certificate will be an authority

####################################################################

[ req ]         # These defaults will be used when making requests to ca
default_bits            = 1024
default_md              = md5
distinguished_name      = req-DN             # defined below
x509_extensions         = dess-si-req-ext    # defined below

[ req-DN ]
countryName                     = Country Name (2 letter code)
countryName_default             = FR
stateOrProvinceName             = State or Province Name (full name)
stateOrProvinceName_default     = France
localityName                    = Locality Name (eg, city)
localityName_default            = Limoges
organizationName                = Organization Name (eg, company)
organizationName_default        = Université de Limoges
organizationalUnitName          = Organisational Unit Name (eg, section)
organizationalUnitName_default  = DESS-SI
commonName                      = Common Name (eg, YOUR name)
emailAddress                    = Email Address

[ dess-si-req-ext ]
basicConstraints = CA:false     # Certificates will not be authorities
```

Je fabrique une paire de clés rsa de 2048 bits pour l'autorité. La clé privée sera dans `private/cakey.pem` et la certificat auto-signé sera dans `cacert.pem`.

```
# openssl req -x509 -config dess-si.cnf -newkey rsa:2048 \
                    -keyout private/cakey.pem -out cacert.pem
```

Je fournis la valeur “Certification Authority” au champ manquant “Common Name” du DN ainsi qu'une passphrase pour chiffrer la clé secrète. Si on veut, on peut faire

```
# openssl x509 -text -noout -in cacert.pem
```

pour examiner le résultat.

Ensuite, vous pouvez créer des clés et faire une requête pour que je la signe (à condition d'avoir un fichier de configuration qui ressemble au mien ou de rajouter des options sur la ligne de commande)

```
$ openssl req -config dess-si.cnf -new -keyout untelkey.pem -out untelreq.pem
```

et m'envoyer le fichier `untelreq.pem` (pas `untelkey.pem` !). Je le signerai par

```
# openssl ca -config dess-si.cnf -in untelreq.pem
```

et je vous renverrai le résultat, ainsi que le certificat de l'autorité. Si j'ai compris quelque chose à tout ça (ce qui est loin d'être sûr), tout navigateur ayant accepté le certificat de l'autorité acceptera automatiquement votre certificat.

4. Tunnels sécurisés

Tout d'abord, une commande qui peut servir de diagnostic :

```
$ openssl s_client -state -connect truffe:443   % Info sur un port géré par SSL
```

Taper 'quit' pour sortir (ou bien des commandes HTTP si ça vous plaît...).

Il peut être intéressant aussi d'examiner le certificat créé lors de l'installation de stunnel

```
$ openssl x509 -text -noout /etc/ssl/stunnel/stunnel.pem
```

HTTPS

J'ai mis un serveur stunnel en écoute sur le port https (443) de truffe par la commande suivante. Le flux de données est donc redirigé sur le port http classique (80).

```
$ stunnel -p /usr/local/stunnel-3.22/stunnel.pem -d https -r localhost:http
```

(il faut aussi contrôler l'accès à `localhost.http` dans `inetd.conf`.)

En faisant une requête sur `https://truffe` avec votre navigateur, on vous proposera d'accepter un certificat que j'ai créé pour l'occasion. Comme je n'ai pas payé ma cotisation à Verisign, ce certificat n'est signé que par lui-même. Mais vous pouvez l'accepter après avoir pris soin de vérifier (en principe) que son empreinte MD-5 est `CC:36:97:00:E2:69:AB:C3:FA:02:B6:13:26:8D:31:18`. Désormais, en vous connectant sur truffe par `https`, ce certificat identifiera truffe. De plus, les échanges HTTP seront chiffrés (à vérifier...).

POP3

J'ai mis un serveur stunnel en écoute sur le port pop3s (995) de truffe par la commande suivante. Lorsque ce port est contacté, le démon `popa3d` est activé (j'aurais pu à la place rediriger vers le port pop3 standard).

```
$ stunnel -p /usr/local/stunnel-3.22/stunnel.pem -d pop3s -l /usr/sbin/popa3d
```

(Il faut aussi contrôler l'accès à `popa3d` dans `/etc/hosts.allow`.)

1. — Sur votre truffe-n locale, lier le port local 10110 au port pop3s de truffe, en mode client (option `-c`) :

```
$ stunnel -c -d 10110 -r truffe:pop3s
```

On peut alors relever son courrier en utilisant le port 10110 local au lieu de truffe:110. La communication est alors chiffrée mais le certificat n'est pas vérifié, donc la sécurité n'est pas clairement améliorée.

2. — Récupérer le certificat `truffe-stunnel.pem` (je l'ai mis sur l'intranet de truffe, dans le répertoire `soft/stunnel`. Installer le sur votre poste, par exemple dans le répertoire `/etc/stunnel/certs` sous le nom `18abd1fe.0` (les 8 chiffres hexadécimaux sont une empreinte du certificat et sont obtenus avec la commande suivante).

```
$ openssl x509 -hash -noout -in truffe-stunnel.pem
```

Ensuite (après avoir fait `killall stunnel` pour détruire le tunnel précédemment créé) vous pouvez remplacer la commande de la question précédente par

```
$ stunnel -c -d 10110 -r truffe:pop3s -a /etc/stunnel/certs -v 2
```

Cette fois ci, le certificat est vérifié et la sécurité est assurée.

5. Bibliographie

Le livre de la collection O'Reilly, "Network security with OpenSSL" par J. Viega, M. Messier et P. Chandra.