Configuration d'Apache *avec modssl †

Rolland Balzon Philippe
Department of Computer Science
SEPRO Robotique
ZI les ajoncs, 85000 La roche sur Yon, France
prolland@free.fr

11 juillet 2002

Résumé

On présente l'ensemble des directives connexes à modssl dans Apache ainsi que leurs impacts.

Table des matières

1 Préambule

Ce rapport a pour but de détailler pas à pas l'installation d'un serveur apache sécurisé via SSL. La description de cette installation, ainsi que les paramétrages qui ont été choisis, l'ont été afin de s'approcher au mieux des besoins de l'entreprise. En particulier, l'authentification de l'utilisateur a été essentiellement priviligiée, par rapport à la sécurité elle-même du site.
Certains paramètres ou détails de configuration peuvent donc nécessiter des changements, si les

*http ://www.apache.org/
†http ://www.modssl.org/

besoins venaient à différer. De ce fait, les paramètres utilisés ont été mis en avant, mais des explications détaillées permettent de changer aisément la configuration en connaissance de cause.

2 Première étape

Une fois l'environnement choisi la première étape consiste à importer les trois fichiers zipés d'Apache, OpenSSL et mod_ssl dans un répertoire source, puis à les déziper. On supposera par la suite que le répertoire source sera */usr/local/*.

Les adresses où récupérer les fichiers sont les suivantes :

http ://www.apache.org/dist/
http ://www.openssl.org/source/
http ://www.modssl.org/source/

Les versions qui ont été utilisées sont apache_1.3.24, openssl-0.9.6d, et mod_ssl-2.8.8-1.3.24.

3 Configuration d'OpenSSL

Tout d'abord, il faut installer OpenSSL. Pour cela il suffit d'accéder au dossier *openssl-0.9.6d*, puis d'effectuer les commandes suivantes :

 # ./Configure
 # make
 # make install

Il faudra alors modifier le fichier de configuration d'OpenSSL (*openssl.cnf*) en fonction des besoins. Normalement, celui-ci se situe dans le dossier */usr/local/ssl/*.

Les principaux paramètres à configurer sont :

Indiquer le répertoire dans lequel seront créés les certificats :
dir=/usr/local/ssl/CA

Indiquer que l'utilisation de certificat se fera pour le transfert de pages chiffrées *www* entre le serveur et le client, mais également pour chiffrer ou signer l'envoi de mails ou d'ojets signés.
nsCertType = client, email, server, objsign

Ensuite se pose la question de la longueur de la clef publique utilisée. Une longueur de 1024 bits serait l'idéal, mais afin d'éviter une trop grande lenteur, les clefs seront choisies de longueur 512 bits :
default_bits = 512

Une dernière partie du fichier *openssl.cnf* reste à modifier en fonction des besoins. C'est ce qui permettra de paramétrer les certificats du serveur et des clients.

Tout d'abord, il faut choisir les informations qui seront nécessaires ou non lors de la création d'un certificat. *match* indique que l'information doit être la même que celle par défaut ; *supplied* indique que cete information est obligatoire (mais pas forcément identique à celle par défaut) ; *optional* indique que cette information n'est pas nécessaire lors de la création d'un certificat.

Dans notre cas on choisira :

policy = policy_match

[policy_match]
countryName = supplied
stateOrProvinceName = supplied
localityName = supplied
organisationName = supplied
organisationUnitName = supplied
commonName = supplied
emailAdress = optional

Ensuite, on pourra indiquer les valeurs que proposera *openssl* lors de la création d'un certificat :

[req_distinguished_name]
countryName_default = FR
0.organizationName_default = SEPRO Robotique

NB : Il est préférable d'éviter les caractères accentués lors de l'écriture des variables par défaut, car certains navigateurs ne les acceptent pas tel quel.

4 Installation de mod_ssl dans Apache

Afin d'installer mod_ssl dans Apache, la première chose à faire après avoir installé Apache est d'accéder au dossier */usr/local/mod_ssl-2.2.8-1.3.24*. La configuration de mod_ssl peut ensuite se faire de différentes façons. Chacune d'elles est décrite dans le fichier *INSTALL* de mod_ssl qu'il est évidemment préférable de lire avant de se lancer. La première façon de procéder : *All-In-One* peut s'avérer plus simple, mais peut-être moins maniable, en particulier si l'on veut ajouter d'autres applications comme mod_php ou mod_perl par la suite.

Un exemple de configuration peut être :

```
# ./Configure     \
        –with-Apache=../Apache_1.3.24      \
        –with-crt=/lien/vers/fichier/server.crt      \
        –with-key=/lien/vers/fichier/server.key
```

NB : Les deux dernières lignes (optionnelles) ne sont à exécuter que si l'on a déjà créé et mis en place les fichiers *server.crt* et *server.key*, ce qui n'est évidemment pas le cas la première fois.

Il faut ensuite retourner dans le dossier */usr/local/Apache_1.3.24*, puis exécuter les commandes suivantes :

```
# SSL_BASE=../openssl-0.9.6d      \
        ./Configure      \
        –enable-module=ssl      \
        –prefix=/usr/local/Apache

# make

# make install
```

Il est à noter que la commande "*make certificate*" peut être effectuée avant le "*make install*". Cependant, cette commande est facultative, et est même déconseillée si lors de l'installation de mod_ssl les options *with-crt* et *with-key* ont été uilisée (cf. ci-dessus).

5 Configuration d'Apache et mod_ssl

La configuration de mod_ssl dans Apache se fait à partir du fichier *httpd.conf* situé dans le dossier Apache. La description qui suit présente les différentes directives qui peuvent être modifiées

lors du paramétrage de mod_ssl.

Il faut savoir que cette description n'est en fait qu'une simple traduction, légèrement modifiée et résumée, de la page de références pour la configuration de mod_ssl dans Apache : *http ://www.modssl.org/docs/2.8/ssl_reference.html*. Il est à noter également que de nombreux termes anglais, n'ayant pas de véritable traduction adéquate en français seront laissés tels quels. Cependant, le paramétrage de la configuration établie sera donné pour chaque directive.

Avant de configurer les différentes directives SSL, il ne faut pas oublier de vérifier que la ligne *AddType application/x-x509-ca-cert .crt* est présente. Sinon, il faut penser à l'ajouter, si l'on veut que le fichier *ca.crt* soit reconnu lors de la configuration du Navigateur. (cf. plus loin).

SSLSessionCache :

Configure le type de stockage du cache *global/inter-process* de la session SSL. Ce cache de session est un service facultatif qui accélère le traitement de demande parallèle. Pour des requêtes au même *server process* (via HTTP *keep-alive*), OpenSSL cache déjà l'information de la session SSL localement. Mais comme les clients peuvent être amenés à charger des images et d'autres données en ligne par l'intermédiaire de demandes parallèles (habituellement jusqu'à quatre demandes parallèles peuvent être communes) ces demandes sont servies par des processus de serveur pré-bifurqués différents. Ici un cache *inter-process* aide à éviter les *handshakes* de session non nécessaires.

Par défaut : pas de cache de session. Cela n'apporte pas d'inconvénients majeurs, si ce n'est une lenteur visiblement accentuée.

dbm : on se sert d'un *DBM hashfile* sur le disque local pour synchroniser les caches locaux de mémoire OpenSSL des processus de serveur. La légère augmentation de I/O sur le serveur a comme conséquence une augmentation de vitesse évidente pour les demandes des clients, ainsi ce type de stockage est généralement recommandé.

Une autre possibilité existe (*shm :/...*) mais elle est relativement plus complexe et n'est pas disponible sur toutes les plates-formes.

Dans notre cas on choisira de préférence :
dbm :/usr/local/apache/logs/ssl_scache

SSLSessionCacheTimeout :

Nombre de secondes avant qu'une session SSL expire dans la session cache. Par défaut elle est configurée à 300 secondes, ce qui peut être gardé tel quel, à moins de vouloir effectuer quelques tests. En ce cas on peut se permettre de descendre jusqu'à 15.

SSLMutex :

Configure le moteur SSL du semaphore employé pour l'exclusion mutuelle des opérations. Cette directive peut être employée seulement dans le contexte du serveur global, car seul un mutex global

est utile.

Par défaut : aucun Mutex. Garder cette configuration par défaut reste risqué, parce qu'actuellement le Mutex est principalement employé pour synchroniser l'accès en écriture à la session cache SSL. Ainsi on ne recommande pas de laisser le Mutex par défaut.

file : C'est la variante portable et toujours fournie (sous Unix) de Mutex où un dossier physique est utilisé comme Mutex. Il est préférable de toujours employer un disque local de système de dossier */path/to/mutex* et non un dossier résidant sur un système de dossier NFS ou AFS. NB : intérieurement, le process ID (PID) du process parent d'Apache est automatiquement apposé au dossier */path/to/mutex* pour le rendre unique, ainsi il n'y a pas à s'inquiéter des conflits. Il est à noter que ce type de mutex n'est pas disponible sous l'environnement Win32. Là on doit employer le mutex de sémaphore.

pem : C'est la plus élégante mais également la moins portative des variantes Mutex où un sémaphore SysV IPC (sous Unix) et un Windows Mutex (sous Win32) est employé quand c'est possible. Elle est seulement disponible quand la plateforme fondamentale la soutient.

Dans notre cas on choisira de préférence :
SSLMutex file :/usr/local/apache/logs/ssl_ mutex

SSLPassPhraseDialog :

Lors du démarrage d'Apache, les fichiers des différents certificats (voir *SSLCertificateFile*) et clefs privées (voir *SSLCertificateKeyFile*) des serveurs virtuels *SSL-enabled* doivent être lus. Puisque pour des raisons de sécurité les fichiers clefs privées sont habituellement chiffrés, mod_ssl doit demander à l'administrateur une *Pass Phrase* (Phrase de passage) pour chacun de ces fichiers.

Les différentes variables sont :

builtin : C'est la variable par défaut. Un dialogue terminal interactif se produit au démarrage. Ici l'administrateur doit écrire manuellement la *Pass Phrase* pour chaque clef privée chiffrée. Comme beaucoup de centres serveurs virtuels *SSL-enabled* peuvent être configurés, l'arrangement suivant est employé pour réduire au minimum le dialogue : quand un fichier clef privée est chiffré, toutes les *Pass Phrase* connues (au début il n'y en a aucune, naturellement) sont essayées. Si l'une d'elle est valable, aucun dialogue n'est requis pour cette clef privée. Si aucune ne correspond, une autre *Pass Phrase* est demandée et est rappelée pour le prochain round (où elle peut être réutilisée).
Cet arrangement permet à mod_ssl d'être flexible au maximum (parce que pour N dossiers clef privée chiffrés, on peut employer N *Pass Phrase* différentes ; l'inconvénient, cependant, c'est qu'il faut rentrer chacune d'entre elle. Le dialogue terminal est malgré tout réduit au minimum (quand on utilise une *Pass Phrase* unique pour toutes les clefs privées, cette *Pass Phrase* est demandée une seule fois (à éviter cependant).

exec :/path/to/program : Ici on configure un programme externe, appelé au démarrage pour chaque fichier clef privée chiffré. Il est appelé avec deux arguments (le premier est de la forme "*servername : portnumber*", le second est "RSA" ou "DSA"), qui indiquent pour quel serveur et algorithme la *Pass Phrase* correspondante doit être affichée au *stdout*. Le principe est que ce programme externe exécute d'abord des contrôles de sécurité pour s'assurer que le système n'est pas compromis par un attaquant, et c'est seulement quand ces contrôles ont été passés avec succès qu'il fournit la *Pass Phrase*.
Ces deux contrôles de sécurité, et la manière dont la *Pass Phrase* est déterminée, peuvent être aussi complexes qu'on le veut. Mod_ssl définit juste l'interface : un exécutable qui fournit la *Pass Phrase* au *stdout*, ni plus ni moins. Ainsi, si l'on est vraiment paranoïaque en terme de sécurité, c'est l'interface qu'il faut.
Il est à noter que le programme externe est appelé seulement une fois par unique *Pass Phrase*.

Laisser *SSLPassPhraseDialog* par défaut s'avère tout à fait suffisant dans notre cas.

SSLRandomSeed :

Cette directive configure une ou plusieurs sources pour semer le générateur de nombre pseudo aléatoire (PRNG) dans OpenSSL au moment du démarrage et/ou juste avant qu'une nouvelle connexion SSL soit établie. Cette directive peut être employée seulement dans le contexte du serveur global parce que le PRNG est un service global.

builtin : C'est la source d'ensemencement *builtin* toujours disponible. Elle utilise les cycles minimum de temps d'exécution CPU et par conséquent peut être toujours employée sans problèmes. L'inconvénient est que ce n'est pas vraiment une source forte, et au démarrage (où le *scoreboard* n'est pas toujours disponible) cette source produit juste quelques bytes d'entropie. Ainsi on devrait toujours, au moins pour le démarrage, employer une source d'ensemencement additionnelle.

file :/path/to/source : Cette variante emploie un dossier externe */path/to/source* comme source pour générer le PRNG. Quand des bytes sont spécifiés, seulement le premier des bytes du dossier forme l'entropie (et les bytes sont donnés dans */path/to/source* comme premier argument). Quand ceux-ci ne sont pas spécifiés, le dossier entier forme l'entropie (et 0 est donné à */path/to/source* comme premier argument). Cette option est particulièrement intéressante quand elle est utilisée au démarrage, par exemple avec des dispositifs disponibles de */dev/random* et/ou */dev/urandom*. Il faut cependant se méfier de cette entropie générée, car lorsqu'il n'y a pas assez d'entropie disponible, on peut se retrouver avec une entropie moins importante que prévue, ou devoir attendre avant d'obtenir une entropie suffisante.

exec :/path/to/program : Cette variante emploie un exécutable externe */path/to/program* comme source pour générer le PRNG. Quand des bytes sont indiqués, seul le premier byte de son contenu *stdout* forme l'entropie. Quand il n'y a pas de bytes de spécifiés, l'intégralité des données produites sur *stdout* forme l'entropie. Cette variante est à employer seulement au moment du démarrage ou l'on a besoin d'un ensemencement très fort avec l'aide d'un programme externe. Employer ceci dans le contexte de raccordement ralentit trop nettement le serveur. Ainsi, habituellement, on se doit d'éviter d'employer des programmes externes dans ce contexte.

egd :/path/to/egd-socket (seulement sous Unix) : Cette variante utilise la douille de domaine d'Unix de l'*Entropy Gathering Daemon* (EGD) externe pour générer le PRNG. A utiliser si aucun dispositif aléatoire n'existe sur la plateforme.

Dans notre cas on choisira de préférence :
SSLRandomSeed startup builtin
SSLRandomSeed connect builtin
Cela permet d'utiliser le minimum de temps d'exécution CPU à chaque connexion, et d'assurer en même temps une entropie suffisante au démarrage.

SSLLog :

Permet de choisir le dossier où écrire le fichier journal consacré au *SSL engine*. Celui-ci doit être choisi dans un endroit où il ne peut pas être employé pour des attaques de type *symlink* sur un vrai serveur (i.e. quelque part où seul l'utilisateur "root" peut écrire).

Dans notre cas on choisira (par exemple) :
SSLLog /usr/local/apache/logs/ssl_engine_log

SSLLogLevel :

Place le degré de verbosité (d'affichage) du fichier journal consacré au protocole *SSL engine Logfile*. Le niveau est l'un des suivant (dans l'ordre croissant) :

none : aucune notation spécifique de SSL n'est faite, mais les messages du type "*error*" sont écrits dans le fichier journal général d'erreur d'Apache.

error : seuls les messages de type "*error*", i.e. les messages qui montrent des situations fatales (processus arrêté) sont affichés. Ces messages sont également reproduits au fichier journal général d'erreur d'Apache.

warn : les messages d'avertissement sont également affichés, i.e. les messages qui montrent des problèmes non fatals (le traitement continue).

info : les messages d'informations sont affichés en plus des précédents, i.e. ceux qui montrent les étapes de transformation principales.

trace : affiche en plus les messages de traces, i.e. les messages qui montrent les étapes de transformation mineures.

debug : les messages de debuggage sont également affichés, i.e. les messages qui montrent le développement et l'information I/O de bas niveau.

Dans notre cas on pourra choisir :

SSLLog info

SSLEngine :

Cette directive permet de basculer (*on/off*) l'utilisation du moteur du protocole de SSL/TLS. Elle est habituellement utilisée à l'intérieur d'une section < VirtualHost > pour permettre SSL/TLS pour un serveur virtuel (*virtual host*) particulier. Par défaut le moteur de protocole de SSL/TLS est handicapé pour le serveur principal et tous les serveurs virtuels configurés.

On choisira donc *SSLEngine on* dans une section *<VirtualHost _default_ :443>*.

SSLCertificateFile :

Permet de pointer vers le fichier *PEM-encoded* du certificat pour le serveur et aussi optionellement vers le fichier de la clef privée RSA ou DSA correspondante (contenue dans le même dossier). Si la clef privée contenue est chiffrée, la "*Pass Phrase dialog*" est obligatoire au démarrage. Cette directive peut être employée jusqu'à deux fois (référençant différents noms de fichier) quand deux certificats de serveur, RSA et DSA sont employés en parallèle.

Dans notre cas on choisira :

SSLCertificateFile /usr/local/apache/conf/ssl.crt/server.crt

SSLCertificateKeyFile :

Pointe vers le fichier *PEM-encoded* de la clef privée pour le serveur. Si la clef privée n'est pas combinée avec le certificat dans le *SSLCertificateFile*, il faut employer cette directive additionnelle pour pointer vers le dossier avec la clef privée autonome (*stand-alone*). Quand *SSLCertificateFile* est employé et que le dossier contient le certificat et la clef privée, cette directive n'a pas besoin

d'être employée. Cependant, cette pratique est fortement déconseillée. Au lieu de cela il est préférable de séparer le certificat et la clef privée. Si la clef privée contenue est chiffrée, la "*Pass Phrase dialog*" est obligatoire au moment du démarrage. Cette directive peut être employée jusqu'à deux fois (référençant différents noms de fichier) quand deux clefs privées, RSA et DSA sont employées en parallèle.

Dans notre cas on choisira :
SSLCertificateKeyFile /usr/local/apache/conf/ssl.key/server.key

SSLCertificateChainFile :

Cette directive permet de placer le dossier (facultatif) complet où l'on peut assembler les certificats des Autorités de Certification (CA) qui forment la chaîne de certificat du certificat du serveur. Un tel dossier est simplement la concaténation des divers dossiers *PEM-encoded* de certificat de CA, habituellement dans l'ordre de chaîne de certificat. Peut être employé alternativement et/ou additionnellement au *SSLCACertificatePath* pour construire explicitement la chaîne de certificat de serveur qui est envoyée au browser en plus du certificat de serveur.

Dans notre cas, il n'est pas nécessaire de l'utiliser.

SSLCACertificatePath :

Place l'annuaire dans lequel on gardera les certificats des Autorités de Certification (CA) des clients que l'on traite. Ils sont employés pour vérifier le certificat du client lors de son authentification.

Dans notre cas on choisira :
SSLCACertificatePath /usr/local/apache/conf/ssl.crt

SSLCACertificateFile :

Place le dossier complet dans lequel on peut assembler les certificats des Autorités de Certification (CA) des clients que l'on a à traiter.

Dans notre cas on choisira :
SSLCACertificateFile /usr/local/apache/conf/ssl.crt/ca.crt

SSLVerifyClient :

Permet de placer le niveau de la vérification de certificat pour l'authentification de client. Il est à noter que cette directive peut être employée dans les deux contextes : par serveur (*per-server*) et par annuaire (*per-directory*). Dans le contexte *per-server* elle s'applique au procédé d'authentification de client utilisé dans le standard SSL *handshake* quand une connexion est établie. Dans le contexte *per-directory* elle force une renégociation SSL avec le niveau de vérification modifié du

client après que la requête HTTP ait été lue mais avant que la réponse HTTP ne soit envoyée.

Les différents niveaux possibles sont :
none : aucun certificat de client n'est exigé (niveau par défaut) ;
optional : le client peut présenter un certificat valide ;
require : le client doit présenter un certificat valide ;
optional_no_ca : le client peut présenter un certificat valide mais il n'a pas besoin d'être vérifiable (avec succès).

Dans notre cas on choisira :
SSLVerifyClient require

SSLVerifyDepth :

Cette directive permet de choisir à quelle "profondeur" mod_ssl doit vérifier avant de décider que les clients n'ont pas un certificat valide. Cette directive peut être utilisée dans les deux contextes par serveur (*per-server*) et par annuaire (*per-directory*). Dans le contexte *per-server* elle s'applique au procédé d'authentification de client utilisé dans le standard SSL *handshake* quand une connexion est établie. Dans le contexte *per-directory* elle force une renégotiation SSL avec la profondeur de vérification (*verification depth*) modifiée du client après que la requête HTTP ait été lue mais avant que la réponse HTTP ne soit envoyée.

La profondeur est en réalité le nombre maximum d'émetteurs intermédiaires de certificat, i.e. le nombre de certificats CA qui sont laissés au maximum pour être suivis tout en vérifiant le certificat du client. Une profondeur de 0 signifie que seuls des certificats auto-signés (*self-signed*) de client sont acceptés (à éviter dans notre cas). La profondeur par défaut de 1 signifie que le certificat du client peut être auto-signé ou doit être signé par une CA qui est directement connue du serveur (i.e. le certificat de CA est sous *SSLCACertificatePath*), etc...

On pourra choisir, par exemple :
SSLVerifyDepth 1

SSLProtocol :

Cette directive peut être employée pour contrôler le choix du protocole SSL que mod_ssl devrait employer en établissant son environnement de serveur. Les clients peuvent alors seulement se relier à un des protocoles fournis.
Les différents protocoles que l'on peut choisir sont *sslv2*, *sslv3*, *tlsv1*, ou *All*. Par défaut, la variable *All* est conservée, ce qui convient tout à fait.

SSLCipherSuite :

Cette directive complexe emploie une chaîne de caractères entrecoupée par des " : ". Elle se compose des caractéristiques du chiffrement OpenSSL pour configurer la suite de chiffrement que le client est autorisé à négocier dans la phase SSL *handshake*. Il est à noter que cette directive peut être employée dans le contexte par serveur (*per-server*) et par annuaire (*per-directory*). Dans le contexte *per-server* elle s'applique au standard SSL *handshake* quand une connexion est éta-

blie. Dans le contexte *per-directory* elle force une renégotiation du protocole SSL avec la suite de chiffrement modifiée, après que la requête HTTP ait été lue mais avant que la réponse HTTP ne soit envoyée. Les spécifications de chiffrement de SSL se composent de 4 attributs principaux plus quelques attributs supplémentaires mineurs.

Les 4 attributs principaux correspondent à :
Algorithme d'échange de clef : RSA ou Diffie-Hellman ;
Algorithme d'authentification : RSA, Diffie-Hellman, DSS ou aucun ;
Algorithme de chiffrement et déchiffrement : DES, Triple-DES, RC4, RC2, IDEA ou aucun ;
Condensé par fonction de hachage (MAC) : MD5, SHA ou SHA1.

Un chiffrement SSL peut également être un chiffrement d'exportation qui est soit *SSLv2* ou soit *SSLv3/TLSv1* (ici *TLSv1* est équivalent à *SSLv3*). Pour indiquer quels chiffrements sont à employer, un peut indiquer tous les chiffrements, un par un, ou employer des alias pour indiquer la préférence et l'ordre pour les chiffrements.

Les alias possibles sont :

Alias	*Description*
$SSLv2$	tous les chiffrements SSL version 2.0
$SSLv3$	tous les chiffrements SSL version 3.0
$TLSv1$	tous les chiffrements TLS version 1.0
EXP	tous les chiffrements d'exportation
$EXPORT40$	tous les chiffrements d'exportation de 40-bit seulement
$EXPORT56$	tous les chiffrements d'exportation de 56-bit seulement
LOW	tous les chiffrements de faible niveau (pas d'exportation, simple DES)
$MEDIUM$	tous les chiffrements avec un codage de 128 bit
$HIGH$	tous les chiffrements utilisant le Triple-DES
RSA	tous les chiffrements utilisant l'échange de clef RSA
DH	tous les chiffrements utilisant l'échange de clef Diffie-Hellman
EDH	tous les chiffrements utilisant l'échange de clef éphémère Diffie-Hellman
ADH	tous les chiffrements utilisant l'échange de clef anonyme Diffie-Hellman
DSS	tous les chiffrements utilisant l'authentification DSS
$NULL$	tous les chiffrements utilisant n'utilisant aucun codage

L'un des intérêts de cette directive est que les variables peuvent servir pour indiquer l'ordre et les chiffrements que l'on souhaite employer. Pour accélérer ceci il existe également les alias *SSLv2, SSLv3, TLSv1, EXP, LOW, MEDIUM, HIGH* (cf. ci-dessus) pour certains groupes de chiffrements. Ces données peuvent être jointes ensembles avec des préfixes pour former le chiffrement spécifié voulu.
Les préfixes possibles sont :
"aucun" : ajoute le(s) chiffrement(s) à la liste ;
+ : ajoute le chiffrement à la liste et le met dans l'endroit courant dans la liste ;
- : enlève le chiffrement de la liste (peut être rajouté par la suite) ;
! : supprime complètement le chiffrement de la liste (ne peut plus être rajouté plus tard) ;

On pourra choisir une *SSLCipherSuite* de la forme :
ALL :!ADH :RC4+RSA :-HIGH :-MEDIUM :-LOW :+EXPORT40 :+SSLv3
Cela permet d'interdire tous les chiffrements qui n'authentifient pas, i.e. seulement le chiffrement Diffie-Hellman anonyme pour le SSL ; cela oblige également SSL à utiliser un chiffrement RC4 avec une clef publique de 40 bits, et un chiffrement asymétrique et une identification RSA.

Il est à noter que la suite proposée par défaut (*ALL :!ADH :RC4+RSA :+HIGH :+MEDIUM :+LOW :+SSLv3 :+EXP*) propose un chiffrement très sécurisé mais qui peut s'avérer trop

lent en exécution.

Afin d'obtenir un gain de temps maximal il est également possible de choisir :

ALL :!ADH :NULL-MD5 :-HIGH :-MEDIUM :-LOW :-EXP :+SSLv3

Lors de la connexion en https, aucun chiffrement ne sera effectué. *NULL-MD5* permet d'autoriser l'absence de chiffrement avec l'utilisation de la fonction de hachage *MD5*, pour l'authentification, qui sera préférée à *SHA1* par exemple qui n'est pas suportée par Netscape et Internet Explorer. Il est également important de rajouter le *+SSLv3*, car *SSLv2* n'accepte pas l'abscence de chiffrement.
Il faut cependant savoir qu'avec cette configuration, même si l'accès est restreint aux utilisateurs pourvus d'un certificat, aucun chiffrement n'est utilisé, et il est donc possible pour une personne écoutant sur la ligne, d'intercepter des informations.

SSLRequire :

Cette directive définit une condition générale qui doit être remplie afin d'autoriser l'accès. C'est une directive très puissante puisque les spécifications de condition sont une expression booléenne arbitrairement complexe permettant le contrôle d'accès.
Notez que l'expression est d'abord analysée dans une représentation interne de machine et ensuite évaluée dans une deuxième étape. En fait, dans le contexte global et celui de classe *per-server*, l'expression est analysée au départ et lors de l'exécution seule la représentation de machine est exécutée. Pour le contexte *per-directory* c'est différent, ici l'expression doit être analysée et immédiatement exécutée pour chaque requête.

Dans notre cas, on pourra choisir, par exemple :

$< Location\ /private >$
$SSLRequire\ \%\{SSL\_CLIENT\_S\_DN\_O\}\ eq\ "Sepro - Robotique"$
$< /Location >$

Il sera également possible (et même recommandé) de rajouter des directives du type :

$\%\{SSL\_CLIENT\_S\_DN\_OU\}\ in\ \{"SAV",\ "Interne",\ ...\}$

Cette directive peut être utilisée dans différentes sous-parties $<$ *Location /souspartie* $>$, afin de restreindre l'accès à des dossiers particuliers à un ou plusieurs groupes.

Ce même système de sous-dossiers peut être utilisé pour la directive précédente (ou pour les autres), afin de sécuriser de façon plus importante certaines sections du site.

Une fois le fichier *httpd.conf* modifié, il faut créer l'Autorité de Certification, puis générer un certificat pour le serveur. En cas de modification ultérieure de ce fichier, l'Autorité de Certification et le certificat pour le serveur restent valide. Il suffit seulement de stoper Apache, puis de le relancer, pour prendre en compte les nouvelles modifications.

6 Création de l'Autorité de Certification

Avant de créer l'Autorité de Certification, il faut se placer dans le dossier */usr/local/ssl*, et y créer un sous-dossier *CA*, s'il n'existe pas déjà. La création de l'Autorité de Certification, ainsi que celles des certificats se feront dans ce sous-dossier.

Les commandes à effectuer sont alors les suivantes :

 # cp /dev/null index.txt

Le fichier *txt* créé est vide au début, mais il contiendra par la suite la description de chacun des certificats qui seront signés.

 # echo 01 > serial

Ce fichier, initialisé à 01 contient le nombre de certificats signés. Ainsi le premier certificat (celui pour le serveur, normalement) aura pour numéro de série 01, le second certificat le numéro 02, etc.

Ensuite, on crée quatre sous répertoires dont on se servira par la suite :

 # mkdir certs crl private newcerts

Enfin, la commande *openssl* va permettre de générer une paire clef privée / clef publique pour l'Autorité de Certification :

 # openssl req -new -x509 -days 1000 -keyout private/cakey.pem -out cacert.pem

La *PEM Phrase* (PEM 1) demandée lors de l'exécution de cette commande sera celle qui permettra par la suite de signer tous les certificats. Il est donc impératif de ne pas l'oublier et de ne pas la divulguer.

7 Création du Certificat pour le Serveur

La création d'un certificat X-509 pour le serveur comprend quatre parties :

1. La création d'une clef privée pour le serveur et la suppression du codage de celle-ci dans le certificat, afin d'éviter qu'Apache nous redemande la phrase de codage à chaque fois qu'on le redémarre.
2. La création du certificat du serveur.
3. La signature de ce certificat par l'Autorité de Certification.
4. L'installation du certificat dans l'environnement du serveur.

Création de la paire de clef :

On se place dans le dossier *private* :
 # cd /usr/local/ssl/CA/private

On exécute alors la commande suivante :
 # openssl genrsa -des3 -out tmp.key 1024

Cette opération va générer une paire clef publique, clef privée à l'aide d'une *PEM Phrase* (PEM 2). Cette paire de clef est alors chiffrée.

 # openssl rsa -in tmp.key -out serverkey.pem

Cette opération demande la phrase de codage du certificat du serveur (PEM 2) et supprime le chiffrement de la clef. Il faut alors protéger ce fichier en lecture seule pour l'administrateur, puis supprimer la clef chiffrée :

 # chmod 400 serverkey.pem
 # rm -f tmp.key

Création du certificat du serveur :

La commande à exécuter est alors la suivante :
 # openssl req -new -days 999 -key serverkey.pem -out newreq.pem

Les informations demandées sont celles choisies dans le fichier *openssl.cnf* et les paramètres rentrés seront visibles lors de l'utilisation de ce certificat sur Internet. Pour que ce certificat soit accepté, il est impératif que tous les champs qui ont été configurés avec la variables *supplied* dans *openssl.cnf* soient remplis. Il est également à noter que la durée de validité de ce certificat doit être inférieure à celle de l'Autorité de Certification.

Signature de ce certificat :

Voici les commandes à effectuer afin de signer le certificat pour le serveur qui vient d'être créé :

 # cd ../certs
 # openssl ca -out servercert.pem -infiles ../private/newreq.pem

La phrase de codage demandée est ici celle de l'autorité de certification (PEM 1).

Installation du certificat :

Pour installer le certificat du serveur dans l'environnement d'Apache, il suffit de le copier dans le dossier correspondant :

 # cd ..
 # cp -p private/serverkey.pem /usr/local/apache/conf/ssl.key/server.key
 # cp -p cacert.pem /usr/local/apache/conf/ssl.crt/ca.crt
 # cp -p certs/servercert.pem /usr/local/apache/conf/ssl.crt/server.crt
 # rm -f private/newreq.pem
 # cd /usr/local/apache/conf/ssl.crt
 # make

Maintenant que ces opérations ont été effectuées, on peut relancer le serveur Apache avec mod_ssl :
 # /usr/local/apache/bin/apachectl startssl

NB : Il est important de relancer apache avec la commande *startssl* et non pas *start* si l'on veut pouvoir se connecter en *https*.

8 Création d'un Certificat pour un Client

Le principe de la création d'un certificat pour un client est sensiblement le même que celui pour le certificat du serveur.

Création du certificat :

La création du certificat se fait de la manière suivante :
 # openssl req -new -keyout newreq.pem -out newreq.pem -days 365

La *PEM Phrase* qui est demandée correspond à celle du certificat du client. Cette *PEM Phrase* ne servira normalement plus, une fois que le certificat du client sera terminé.
De même que pour le certificat du serveur, tous les champs obligatoires (*supplied*) doivent être remplis. Pour les champs facultatifs, on peut entrer un "." pour indiquer une valeur nulle.

Signature du certificat :

Pour signer la requête de certificat, il suffit, comme pour le certificat du serveur, d'exécuter la commande suivante :
 # openssl ca -out newcert.pem -infiles newreq.pem
La *PEM Phrase* demandée est alors la phrase de l'Autorité de Certification (PEM 1).

Exportation du certificat :

Le fichier résultat de ces deux opérations n'est en général pas utilisable directement. Il faut donc l'exporter au format *pkcs12* :
 # openssl pkcs12 -export -in newcert.pem -inkey newreq.pem -name "Prénom Nom" -clcerts -out "nom.p12"

NB : *Prénom Nom* sera le nom du certificat ; le fichier obtenu, lui, aura pour nom *nom.p12*. Le mot de passe de protection demandé lors de l'exportation sera indispensable à l'implémentation du certificat dans le navigateur par la suite. Il doit être transmis par un moyen sûr, en même temps que le certificat.

9 Configuration du Navigateur

L'importation du certificat d'un utilisateur doit se faire de manière totalement sûre. Pour cela, il y a deux façon de faire : soit l'administateur importe lui même le certificat pour chaque utilisateur (coûteux en temps), soit chaque utilisateur reçoit son certificat sur une disquette, avec le mot de passe d'exportation correspondant, ainsi qu'un guide explicatif d'installation, et doit alors installer lui-même son certificat.

Configuration de Netscape :

La version de Netscape utilisée lors de la configuration est celle de *Netscape Communicator 4.74*, mais le principe reste le même pour les autres versions, ainsi que pour Internet Explorer.

La première chose à faire est de lancer Netscape et d'accéder à l'URL du fichier *ca.crt*. Une série d'avertissements apparaît pour demander si l'on désire accepter cette nouvelle Autorité de Certification. Il suffit alors de cliquer sur le bouton *suivant*, jusqu'à ce qu'apparaisse le tableau demandant pour quels types de certification on souhaite utiliser ce nouveau certificat. Cocher les trois cases permet de l'utiliser pour tout type de service, avant de cliquer à nouveau sur *suivant*. Enfin, un tableau apparaît, demandant le nom de cette Autorité de Certification.

Pour vérifier que cette Autorité de Certification a été aceptée, il suffit de cliquer sur l'icône *security* de Netscape, puis d'aller dans la rubrique *signers*. Normalement la nouvelle Autorité de Certification doit se trouver dans la liste. On peut alors la sélectionner et cliquer sur le bouton *Verify*. Le Navigateur indique alors si elle a été vérifiée avec succès.

NB : Pour que cette configuration puisse être effectuée sans problèmes, il faut que l'accès au fichier *ca.crt* se fasse en https, ou au moins que la commande *AddType application/x-x509-ca-cert .crt* soit présente dans le fichier *httpd.conf*, afin que le fichier *ca.crt* soit reconnu.

Importation du certificat :

L'importation du certificat doit se faire avant d'accéder au serveur sécurisé. Pour cela il faut, dans Netscape, cliquer sur l'icône *security* ou sélectionner le menu *Communicator* puis *Tools*, et enfin *Security Info*. Une invite de commande apparaît. Dans le menu à droite pointez sur le lien *Yours* (en-dessous de *Certificates*) et cliquez sur le bouton *Import a Certificate*. Il n'y a alors plus qu'à aller chercher le certificat voulu. A ce moment, le navigateur demande d'entrer le mot de passe protégeant l'exportation du certificat. Ce mot de passe est celui qui a été choisi lors de l'exportation du certificat au format pkcs12.

Il est alors possible de visualiser les informations de ce certificat en le sélectionnant et en cliquant sur *View*.

NB : Il n'est normalement pas nécessaire de conserver le fichier p12 sur le disque dur. Dans le cas d'un certificat importé sur une disquette, il est même possible et préférable d'importer ce certificat directement de la disquette puis de la détruire par la suite, ou de la renvoyer à l'Autorité de Certification.

Un autre paramétrage est à vérifier, en particulier si on effectue une authentification seule sans chiffrement, il s'agit de choisir quelle configuration SSL on autorise pour le navigateur. Dans la fenêtre *security* toujours, cliquer sur le lien *Navigator*. Il y a alors plusieurs paramètres à choisir :

Tout d'abord, choisir pour l'accès à quel type de page on veut recevoir un avertissement.

Ensuite, choisir dans le menu déroulant quel certificat est uilisé pour l'authentification. Il est préférable d'éviter de choisir *Ask Every Time* si l'on ne veut pas que le Navigateur redemande lors de chaque connexion le choix du certificat ; au contraire, choisir *Select Automatically* est beaucoup plus adapté. Imposer le certificat que l'on vient d'importer peut s'avérer limité si l'utilisateur possède plusieurs certificats.

Enfin, le dernier paramétrage à vérifier concerne celui du chiffrement utilisé. Il est préférable de cocher les deux cases afin d'autoriser SSLv2 et SSLv3, pour ne pas limiter le navigateur, et surtout il est important de vérifier dans la configuration de SSLv3 que la case *No encryption with an MD5 MAC* est cochée. Sinon le navigateur n'acceptera pas de se connecter à une page *https* qui ne soit pas chiffrée. Il est également préférable d'autoriser également les autres chiffrements, si l'on veut pouvoir se connecter sur d'autres sites sécurisés.

10 Signer ou chiffrer des mails

Signer un message :

Pour signer un message avec son certificat, il suffit à l'utilisateur de rédiger son message, puis de cliquer sur l'icône *security* dans la barre de menu de son message, et de cocher la case située sous *Signing Message*.

Le destinataire recevra alors le message avec une icône, indiquant que le message a été correctement signé, si le certificat correspond bien à celui de l'expéditeur. Sinon, cette icône indique que la signature n'est pas valide.

Chiffrer un message :

Pour chiffrer un message, le principe est exactement le même, sauf qu'il faut cliquer sur la case située sous *Encrypting Message*.

Il est à noter cependant qu'il n'est possible de chiffrer ce message que si Netscape connaît le certificat du destinataire. Pour cela il faut avoir déjà reçu un message signé de ce destinataire, ou bien que chacun des certificats soient disponibles dans un annuaire (base LDAP par exemple).

Cela dit, cet annuaire n'est pas forcément nécessaire, puisque Netscape est capable de mémoriser lui-même les certificats des autres utilisateurs. Pour cela, il faut cliquer sur l'icône de signature (en haut à droite dans Netscape Messenger), une fois que l'on a reçu un message signé. Une fenêtre de dialogue apparaît alors, détaillant le certificat de l'utilisateur. Le certificat de l'expéditeur est alors mémorisé automatiquement et sera visible dans la section *people*.

11 Références

Les pages du manuel d'utilisation de mod_ssl :
http ://www.modssl.org/docs/2.8/

Description détaillée du principe du protocole SSL :
http ://developer.netscape.com/docs/manuals/proxy/adminux/encrypt.htm

Description du principe du protocole SSL :
http ://www.guill.net/reseaux/Ssl.html

Rappel des principes du protocole SSL et gestion des certificats :
http ://www.althes.fr/Technologie/Authentcertif.htm

Deux pages de description pas à pas de la mise en oeuvre de SSL :

Celle réalisée par la Délégation Poitou-Charentes du CNRS :
http ://www.dr15.cnrs.fr/Delegation/STI/Certifs/guide/

Celle réalisée par l'université de Jussieu, qui présente également la mise en place de imaps :
http ://www.math.jussieu.fr/ jas/imap.html