Serveur HTTP Apache Version 2.4

Chiffrement SSL/TLS fort: foire aux questions

Le sage n'apporte pas de bonnes réponses, il pose les bonnes questions
-- Claude Levi-Strauss

Installation

Pourquoi le démarrage d'Apache provoque-t-il des erreurs de permission en rapport avec SSLMutex ?

Des erreurs telles que ``mod_ssl: Child could not open SSLMutex lockfile /opt/apache/logs/ssl_mutex.18332 (avec l'erreur système qui suit) [...] System: Permission denied (errno: 13)" sont souvent provoquées par des permissions trop restrictives sur les répertoires *parents*. Assurez-vous que tous les répertoires parents (ici `/opt`, `/opt/apache` et `/opt/apache/logs`) ont le bit x positionné au moins pour l'UID sous lequel les processus enfants d'Apache s'exécutent (voir la directive `User`).

Pourquoi mod_ssl s'arrête-t-il avec l'erreur "Failed to generate temporary 512 bit RSA private key" au démarrage d'Apache ?

Pour fonctionner correctement, les logiciels de cryptographie ont besoin d'une source de données aléatoires. De nombreux systèmes d'exploitation libres proposent un "périphérique source d'entropie" qui fournit ce service (il se nomme en général `/dev/random`). Sur d'autres systèmes, les applications doivent amorcer manuellement le Générateur de Nombres Pseudo-Aléatoires d'OpenSSL (Pseudo Random Number Generator -PRNG) à l'aide de données appropriées avant de générer des clés ou d'effectuer un chiffrement à clé publique. Depuis la version 0.9.5, les fonctions d'OpenSSL qui nécessitent des données aléatoires provoquent une erreur si le PRNG n'a pas été amorcé avec une source de données aléatoires d'au moins 128 bits.

Pour éviter cette erreur, `mod_ssl` doit fournir suffisamment d'entropie au PRNG pour lui permettre de fonctionner correctement. Ce niveau d'entropie est défini par la directive `SSLRandomSeed`.

Configuration

Peut-on faire cohabiter HTTP et HTTPS sur le même serveur ?

Oui. HTTP et HTTPS utilisent des ports différents (HTTP écoute le port 80 et HTTPS le port 443), si bien qu'il n'y a pas de conflit direct entre les deux. Vous pouvez soit exécuter deux instances séparées du serveur, chacune d'entre elles écoutant l'un de ces ports, soit utiliser l'élégante fonctionnalité d'Apache que constituent les hôtes virtuels pour créer deux serveurs virtuels gérés par la même instance d'Apache - le premier serveur répondant en HTTP aux requêtes sur le port 80, le second répondant en HTTPS aux requêtes sur le port 443.

Quel port HTTPS utilise-t-il ?

Vous pouvez associer le protocole HTTPS à n'importe quel port, mais le port standard est le port 443, que tout navigateur compatible HTTPS va utiliser par défaut. Vous pouvez forcer votre navigateur à utiliser un port différent en le précisant dans l'URL. Par exemple, si votre serveur est configuré pour servir des pages en HTTPS sur le port 8080, vous pourrez y accéder par l'adresse `https://example.com:8080/`.

Comment s'exprimer en langage HTTPS à des fins de test ?

Alors que vous utilisez simplement

```
$ telnet localhost 80
GET / HTTP/1.0
```

pour tester facilement Apache via HTTP, les choses ne sont pas si simples pour HTTPS à cause du protocole SSL situé entre TCP et HTTP. La commande OpenSSL `s_client` vous permet cependant d'effectuer un test similaire via HTTPS :

```
$ openssl s_client -connect localhost:443 -state -debug
GET / HTTP/1.0
```

Avant la véritable réponse HTTP, vous recevrez des informations détaillées à propos de l'établissement de la connexion SSL. Si vous recherchez un client en ligne de commande à usage plus général qui comprend directement HTTP et HTTPS, qui peut effectuer des opérations GET et POST, peut utiliser un mandataire, supporte les requêtes portant sur une partie d'un fichier (byte-range), etc..., vous devriez vous tourner vers l'excellent outil cURL (↗ http://curl.haxx.se/) . Grâce à lui, vous pouvez vérifier si Apache répond correctement aux requêtes via HTTP et HTTPS comme suit :

```
$ curl http://localhost/
$ curl https://localhost/
```

Pourquoi la communication se bloque-t-elle lorsque je me connecte à mon serveur Apache configuré pour SSL ?

Ceci peut arriver si vous vous connectez à un serveur HTTPS (ou à un serveur virtuel) via HTTP (par exemple, en utilisant `http://example.com/` au lieu de `https://example.com`). Cela peut aussi arriver en essayant de vous connecter via HTTPS à un serveur HTTP (par exemple, en utilisant `https://example.com/` avec un serveur qui ne supporte pas HTTPS, ou le supporte, mais sur un port non standard). Assurez-vous que vous vous connectez bien à un serveur (virtuel) qui supporte SSL.

Pourquoi, lorsque je tente d'accéder en HTTPS à mon serveur Apache+mod_ssl fraîchement installé, l'erreur ``Connection Refused'' s'affiche-t-elle ?

Une configuration incorrecte peut provoquer ce type d'erreur. Assurez-vous que vos directives `Listen` s'accordent avec vos directives `<VirtualHost>`. Si l'erreur persiste, recommencez depuis le début en restaurant la configuration par défaut fournie par`mod_ssl`.

Pourquoi les variables `SSL_XXX` ne sont-elles pas disponibles dans mes scripts CGI et SSI ?

Assurez-vous que la directive ``SSLOptions +StdEnvVars'' est bien présente dans le contexte de vos requêtes CGI/SSI.

Comment puis-je basculer entre les protocoles HTTP et HTTPS dans les hyperliens relatifs ?

Normalement, pour basculer entre HTTP et HTTPS, vous devez utiliser des hyperliens pleinement qualifiés (car vous devez modifier le schéma de l'URL). Cependant, à l'aide du module `mod_rewrite`, vous pouvez manipuler des hyperliens relatifs, pour obtenir le même effet.

```
RewriteEngine on
RewriteRule    "^/(.*)_SSL$"    "https://%{SERVER_NAME}/$1" [R,L]
RewriteRule    "^/(.*)_NOSSL$" "http://%{SERVER_NAME}/$1"  [R,L]
```

Ce jeu de règles rewrite vous permet d'utiliser des hyperliens de la forme `<a href="document.html_SSL">` pour passer en HTTPS dans les liens relatifs. (Remplacez SSL par NOSSL pour passer en HTTP.)

Certificats

- Qu'est-ce qu'un clé privée RSA, un certificat, une demande de signature de certificat (CSR) ?
- Y a-t-il une différence au démarrage entre un serveur Apache non SSL et un serveur Apache supportant SSL ?
- Comment créer un certificat auto-signé SSL à des fins de test ?
- Comment créer un vrai certificat SSL ?
- Comment créer et utiliser sa propre Autorité de certification (CA) ?
- Comment modifier le mot de passe de ma clé privée ?

- Comment démarrer Apache sans avoir à entrer de mot de passe ?
- Comment vérifier si une clé privée correspond bien à son certificat ?
- Comment convertir un certificat du format PEM au format DER ?
- Pourquoi les navigateurs se plaignent-ils de ne pas pouvoir vérifier mon certificat de serveur ?

Qu'est-ce qu'un clé privée RSA, un certificat, une demande de signature de certificat (CSR) ?

Un fichier de clé privée RSA est un fichier numérique que vous pouvez utiliser pour déchiffrer des messages que l'on vous a envoyés. Il a son pendant à caractère public que vous pouvez distribuer (par le biais de votre certificat), ce qui permet aux utilisateurs de chiffrer les messages qu'ils vous envoient.

Une Demande de Signature de Certificat (CSR) est un fichier numérique qui contient votre clé publique et votre nom. La CSR doit être envoyée à une Autorité de Certification (CA), qui va la convertir en vrai certificat en la signant.

Un certificat contient votre clé publique RSA, votre nom, le nom de la CA, et est signé numériquement par cette dernière. Les navigateurs qui reconnaissent la CA peuvent vérifier la signature du certificat, et ainsi en extraire votre clé publique RSA. Ceci leur permet de vous envoyer des messages chiffrés que vous seul pourrez déchiffrer.

Se référer au chapitre Introduction (↗ ssl_intro.html) pour une description générale du protocole SSL.

Y a-t-il une différence au démarrage entre un serveur Apache non SSL et un serveur Apache supportant SSL ?

Oui. En général, avec ou sans `mod_ssl` intégré, le démarrage d'Apache ne présente pas de différences. Cependant, si votre fichier de clé privée SSL possède un mot de passe, vous devrez le taper au démarrage d'Apache.

Devoir entrer manuellement le mot de passe au démarrage du serveur peut poser quelques problèmes - par exemple, quand le serveur est démarré au moyen de scripts au lancement du système. Dans ce cas, vous pouvez suivre les étapes ci-dessous (↗ #removepassphrase) pour supprimer le mot de passe de votre clé privée. Gardez à l'esprit qu'agir ainsi augmente les risques de sécurité - agissez avec précaution !

Comment créer un certificat auto-signé SSL à des fins de test ?

1. Vérifiez qu'OpenSSL est installé et l'exécutable openssl dans votre `PATH`.

2. Exécuter la commande suivante pour créer les fichiers `server.key` et `server.crt` :
   `$ openssl req -new -x509 -nodes -out server.crt -keyout server.key`
   Ces fichiers seront utilisés comme suit dans votre `httpd.conf` :

   ```
   SSLCertificateFile    "/path/to/this/server.crt"
   SSLCertificateKeyFile "/path/to/this/server.key"
   ```

3. Il est important de savoir que le fichier `server.key` n'a *pas* de mot de passe. Pour ajouter un mot de passe à la clé, vous devez exécuter la commande suivante et confirmer le mot de passe comme demandé.
   `$ openssl rsa -des3 -in server.key -out server.key.new`
   `$ mv server.key.new server.key`

   Sauvegardez le fichier `server.key` ainsi que son mot de passe en lieu sûr.

Comment créer un vrai certificat SSL ?

Voici la marche à suivre pas à pas :

1. Assurez-vous qu'OpenSSL est bien installé et dans votre `PATH`.

2. Créez une clé privée RSA pour votre serveur Apache (elle sera au format PEM et chiffrée en Triple-DES):

   `$ openssl genrsa -des3 -out server.key 2048`

   Enregistrez le fichier `server.key` et le mot de passe éventuellement défini en lieu sûr. Vous pouvez afficher les détails de cette clé privée RSA à l'aide de la commande :

   `$ openssl rsa -noout -text -in server.key`

   Si nécessaire, vous pouvez aussi créer une version PEM non chiffrée (non recommandé) de clé privée RSA avec :

   `$ openssl rsa -in server.key -out server.key.unsecure`

3. Créez une Demande de signature de Certificat (CSR) à l'aide de la clé privée précédemment générée (la sortie sera au format PEM):

```
$ openssl req -new -key server.key -out server.csr
```

Vous devez entrer le Nom de Domaine Pleinement Qualifié ("Fully Qualified Domain Name" ou FQDN) de votre serveur lorsqu'OpenSSL vous demande le "CommonName", c'est à dire que si vous générez une CSR pour un site web auquel on accèdera par l'URL `https://www.foo.dom/`, le FQDN sera "www.foo.dom". Vous pouvez afficher les détails de ce CSR avec :

```
$ openssl req -noout -text -in server.csr
```

4. Vous devez maintenant envoyer la CSR à une Autorité de Certification (CA), afin que cette dernière puisse la signer. Une fois la CSR signée, vous disposerez d'un véritable certificat que vous pourrez utiliser avec Apache. Vous pouvez faire signer votre CSR par une CA commerciale ou par votre propre CA.
   Les CAs commerciales vous demandent en général de leur envoyer la CSR par l'intermédiaire d'un formulaire web, de régler le montant de la signature, puis vous envoient un certificat signé que vous pouvez enregistrer dans un fichier server.crt. Pour plus de détails sur la manière de créer sa propre CA, et de l'utiliser pour signer une CSR, voir ci-dessous.
   Une fois la CSR signée, vous pouvez afficher les détails du certificat comme suit :

   ```
   $ openssl x509 -noout -text -in server.crt
   ```

5. Vous devez maintenant disposer de deux fichiers : `server.key` et `server.crt`. Ils sont précisés dans votre fichier `httpd.conf` comme suit :

   ```
   SSLCertificateFile    "/path/to/this/server.crt"
   SSLCertificateKeyFile "/path/to/this/server.key"
   ```

   Le fichier `server.csr` n'est plus nécessaire.

Comment créer et utiliser sa propre Autorité de certification (CA) ?

La solution la plus simple consiste à utiliser les scripts `CA.sh` ou `CA.pl` fournis avec OpenSSL. De préférence, utilisez cette solution, à moins que vous ayez de bonnes raisons de ne pas le faire. Dans ce dernier cas, vous pouvez créer un certificat auto-signé comme suit :

1. Créez une clé privée RSA pour votre serveur (elle sera au format PEM et chiffrée en Triple-DES) :

   ```
   $ openssl genrsa -des3 -out server.key 2048
   ```

   Sauvegardez le fichier `server.key` et le mot de passe éventuellement défini en lieu sûr. Vous pouvez afficher les détails de cette clé privée RSA à l'aide de la commande :

   ```
   $ openssl rsa -noout -text -in server.key
   ```

   Si nécessaire, vous pouvez aussi créer une version PEM non chiffrée (non recommandé) de cette clé privée RSA avec :

   ```
   $ openssl rsa -in server.key -out server.key.unsecure
   ```

2. Créez un certificat auto-signé (structure X509) à l'aide de la clé RSA que vous venez de générer (la sortie sera au format PEM) :

   ```
   $ openssl req -new -x509 -nodes -sha1 -days 365 -key server.key -out server.crt
   -extensions usr_cert
   ```

   Cette commande signe le certificat du serveur et produit un fichier `server.crt`. Vous pouvez afficher les détails de ce certificat avec :

   ```
   $ openssl x509 -noout -text -in server.crt
   ```

Comment modifier le mot de passe de ma clé privée ?

Vous devez simplement lire la clé avec l'ancien mot de passe et la réécrire en spécifiant le nouveau mot de passe. Pour cela, vous pouvez utiliser les commandes suivantes :

```
$ openssl rsa -des3 -in server.key -out server.key.new
$ mv server.key.new server.key
```

La première fois qu'il vous est demandé un mot de passe PEM, vous devez entrer l'ancien mot de passe. Ensuite, on vous demandera d'entrer encore un mot de passe - cette fois, entrez le nouveau mot de passe. Si on vous demande de vérifier le mot de passe, vous devrez entrer le nouveau mot de passe une seconde fois.

Comment démarrer Apache sans avoir à entrer de mot de passe ?

L'apparition de ce dialogue au démarrage et à chaque redémarrage provient du fait que la clé privée RSA contenue dans votre fichier

server.key est enregistrée sous forme chiffrée pour des raisons de sécurité. Le déchiffrement de ce fichier nécessite un mot de passe, afin de pouvoir être lu et interprété. Cependant, La suppression du mot de passe diminue le niveau de sécurité du serveur - agissez avec précautions !

1. Supprimer le chiffrement de la clé privée RSA (tout en conservant une copie de sauvegarde du fichier original) :

```
$ cp server.key server.key.org
$ openssl rsa -in server.key.org -out server.key
```

2. Assurez-vous que le fichier server.key n'est lisible que par root :

```
$ chmod 400 server.key
```

Maintenant, `server.key` contient une copie non chiffrée de la clé. Si vous utilisez ce fichier pour votre serveur, il ne vous demandera plus de mot de passe. CEPENDANT, si quelqu'un arrive à obtenir cette clé, il sera en mesure d'usurper votre identité sur le réseau. Vous DEVEZ par conséquent vous assurer que seuls root ou le serveur web peuvent lire ce fichier (de préférence, démarrez le serveur web sous root et faites le s'exécuter sous un autre utilisateur, en n'autorisant la lecture de la clé que par root).

Une autre alternative consiste à utiliser la directive ``SSLPassPhraseDialog exec:/chemin/vers/programme". Gardez cependant à l'esprit que ce n'est bien entendu ni plus ni moins sécurisé.

Comment vérifier si une clé privée correspond bien à son certificat ?

Une clé privée contient une série de nombres. Deux de ces nombres forment la "clé publique", les autres appartiennent à la "clé privée". Les bits de la "clé publique" sont inclus quand vous générez une CSR, et font par conséquent partie du certificat associé.

Pour vérifier que la clé publique contenue dans votre certificat correspond bien à la partie publique de votre clé privée, il vous suffit de comparer ces nombres. Pour afficher le certificat et la clé, utilisez cette commande :

```
$ openssl x509 -noout -text -in server.crt
$ openssl rsa -noout -text -in server.key
```

Les parties `modulus' et `public exponent' doivent être identiques dans la clé et le certificat. Comme le `public exponent' est habituellement 65537, et comme il est difficile de vérifier visuellement que les nombreux nombres du `modulus' sont identiques, vous pouvez utiliser l'approche suivante :

```
$ openssl x509 -noout -modulus -in server.crt | openssl md5
$ openssl rsa -noout -modulus -in server.key | openssl md5
```

Il ne vous reste ainsi que deux nombres relativement courts à comparer. Il est possible, en théorie que ces deux nombres soient les mêmes, sans que les nombres du modulus soient identiques, mais les chances en sont infimes.

Si vous souhaitez vérifier à quelle clé ou certificat appartient une CSR particulière, vous pouvez effectuer le même calcul sur la CSR comme suit :

```
$ openssl req -noout -modulus -in server.csr | openssl md5
```

Comment convertir un certificat du format PEM au format DER ?

Le format des certificats par défaut pour OpenSSL est le format PEM, qui est tout simplement un format DER codé en Base64, avec des lignes d'en-têtes et des annotations. Certaines applications, comme Microsoft Internet Explorer, ont besoin d'un certificat au format DER de base. Vous pouvez convertir un fichier PEM `cert.pem` en son équivalent au format DER `cert.der` à l'aide de la commande suivante : **`$ openssl x509 -in cert.pem -out cert.der -outform DER`**

Pourquoi les navigateurs se plaignent-ils de ne pas pouvoir vérifier mon certificat de serveur ?

Ceci peut se produire si votre certificat de serveur est signé par une autorité de certification intermédiaire. Plusieurs CAs, comme Verisign ou Thawte, ont commencé à signer les certificats avec des certificats intermédiaires au lieu de leur certificat racine.

Les certificats de CA intermédiaires se situe à un niveau intermédiaire entre le certificat racine de la CA (qui est installé dans les navigateurs) et le certificat du serveur (que vous avez installé sur votre serveur). Pour que le navigateur puisse traverser et vérifier la chaîne de confiance depuis le certificat du serveur jusqu'au certificat racine, il faut lui fournir les certificats intermédiaires. Les CAs devraient pouvoir fournir de tels paquetages de certificats intermédiaires à installer sur les serveurs.

Vous devez inclure ces certificats intermédiaires via la directive `SSLCertificateChainFile`.

Le protocole SSL

- Pourquoi de nombreuses et aléatoires erreurs de protocole SSL apparaissent-elles en cas de forte charge du serveur ?
- Pourquoi la charge de mon serveur est-elle plus importante depuis qu'il sert des ressources chiffrées en SSL ?
- Pourquoi les connexions en HTTPS à mon serveur prennent-elles parfois jusqu'à 30 secondes pour s'établir ?

Pourquoi de nombreuses et aléatoires erreurs de protocole SSL apparaissent-elles en cas de forte charge du serveur ?

Ce problème peut avoir plusieurs causes, mais la principale réside dans le cache de session SSL défini par la directive `SSLSessionCache`. Le cache de session DBM est souvent à la source du problème qui peut être résolu en utilisant le cache de session SHM (ou en n'utilisant tout simplement pas de cache).

Pourquoi la charge de mon serveur est-elle plus importante depuis qu'il sert des ressources chiffrées en SSL ?

SSL utilise un procédé de chiffrement fort qui nécessite la manipulation d'une quantité très importante de nombres. Lorsque vous effectuez une requête pour une page web via HTTPS, tout (même les images) est chiffré avant d'être transmis. C'est pourquoi un accroissement du traffic HTTPS entraîne une augmentation de la charge.

Pourquoi les connexions en HTTPS à mon serveur prennent-elles parfois jusqu'à 30 secondes pour s'établir ?

Ce problème provient en général d'un périphérique `/dev/random` qui bloque l'appel système read(2) jusqu'à ce que suffisamment d'entropie soit disponible pour servir la requête. Pour plus d'information, se référer au manuel de référence de la directive `SSLRandomSeed`.

Quels sont les algorithmes de chiffrement supportés par mod_ssl ?

En général, tous les algorithmes de chiffrement supportés par la version d'OpenSSL installée, le sont aussi par `mod_ssl`. La liste des algorithmes disponibles peut dépendre de la manière dont vous avez installé OpenSSL. Typiquement, au moins les algorithmes suivants sont supportés :

1. RC4 avec SHA1
2. AES avec SHA1
3. Triple-DES avec SHA1

Pour déterminer la liste réelle des algorithmes disponibles, vous pouvez utiliser la commande suivante :

```
$ openssl ciphers -v
```

Pourquoi une erreur ``no shared cipher'' apparaît-elle quand j'essaie d'utiliser un algorithme de chiffrement Diffie-Hellman anonyme (ADH) ?

Par défaut et pour des raisons de sécurité, OpenSSl ne permet *pas* l'utilisation des algorithmes de chiffrements ADH. Veuillez vous informer sur les effets pervers potentiels si vous choisissez d'activer le support de ces algorithmes de chiffrements.

Pour pouvoir utiliser les algorithmes de chiffrements Diffie-Hellman anonymes (ADH), vous devez compiler OpenSSL avec ``-`DSSL_ALLOW_ADH`'', puis ajouter ```ADH`'' à votre directive `SSLCipherSuite`.

Pourquoi une erreur ``no shared cipher'' apparaît-elle lorsqu'on se connecte à mon serveur fraîchement installé ?

Soit vous avez fait une erreur en définissant votre directive `SSLCipherSuite` (comparez-la avec l'exemple préconfiguré dans `extra/httpd-ssl.conf`), soit vous avez choisi d'utiliser des algorithmes DSA/DH au lieu de RSA lorsque vous avez généré votre clé privée, et avez ignoré ou êtes passé outre les avertissements. Si vous avez choisi DSA/DH, votre serveur est incapable de communiquer en utilisant des algorithmes de chiffrements SSL basés sur RSA (du moins tant que vous n'aurez pas configuré une paire clé/certificat RSA additionnelle). Les navigateurs modernes tels que NS ou IE ne peuvent communiquer par SSL qu'avec des algorithmes RSA. C'est ce qui provoque l'erreur "no shared ciphers". Pour la corriger, générez une nouvelle paire clé/certificat pour le serveur en utilisant un algorithme de chiffrement RSA.

Pourquoi ne peut-on pas utiliser SSL avec des hôtes virtuels identifiés par un nom et non par une adresse IP ?

La raison est très technique, et s'apparente au problème de la primauté de l'oeuf ou de la poule. La couche du protocole SSL se trouve en dessous de la couche de protocole HTTP qu'elle encapsule. Lors de l'établissement d'une connexion SSL (HTTPS), Apache/mod_ssl doit négocier les paramètres du protocole SSL avec le client. Pour cela, mod_ssl doit consulter la configuration du serveur virtuel (par exemple, il doit accéder à la suite d'algorithmes de chiffrement, au certificat du serveur, etc...). Mais afin de sélectionner le bon serveur virtuel, Apache doit connaître le contenu du champ d'en-tête HTTP `Host`. Pour cela, il doit lire l'en-tête de la requête HTTP. Mais il ne peut le faire tant que la négociation SSL n'est pas terminée, or, la phase de négociation SSL a besoin du nom d'hôte contenu dans l'en-tête de la requête. Voir la question suivante pour contourner ce problème.

Notez que si votre certificat comporte un nom de serveur avec caractères génériques, ou des noms de serveurs multiples dans le champ subjectAltName, vous pouvez utiliser SSL avec les serveurs virtuels à base de noms sans avoir à contourner ce problème.

Est-il possible d'utiliser l'hébergement virtuel basé sur le nom d'hôte pour différencier plusieurs hôtes virtuels ?

L'hébergement virtuel basé sur le nom est une méthode très populaire d'identification des différents hôtes virtuels. Il permet d'utiliser la même adresse IP et le même numéro de port pour de nombreux sites différents. Lorsqu'on se tourne vers SSL, il semble tout naturel de penser que l'on peut appliquer la même méthode pour gérer plusieurs hôtes virtuels SSL sur le même serveur.

C'est possible, mais seulement si on utilise une version 2.2.12 ou supérieure du serveur web compilée avec OpenSSL version 0.9.8j ou supérieure. Ceci est du au fait que l'utilisation de l'hébergement virtuel à base de nom avec SSL nécessite une fonctionnalité appelée Indication du Nom de Serveur (Server Name Indication - SNI) que seules les révisions les plus récentes de la spécification SSL supportent.

Notez que si votre certificat comporte un nom de serveur avec caractères génériques, ou des noms de serveurs multiples dans le champ subjectAltName, vous pouvez utiliser SSL avec les serveurs virtuels à base de noms sans avoir à contourner ce problème.

La raison en est que le protocole SSL constitue une couche séparée qui encapsule le protocole HTTP. Aini, la session SSL nécessite une transaction séparée qui prend place avant que la session HTTP n'ait débuté. Le serveur reçoit une requête SSL sur l'adresse IP X et le port Y (habituellement 443). Comme la requête SSL ne contenait aucun en-tête Host:, le serveur n'avait aucun moyen de déterminer quel hôte virtuel SSL il devait utiliser. En général, il utilisait le premier qu'il trouvait et qui correspondait à l'adresse IP et au port spécifiés.

Par contre, si vous utilisez des versions du serveur web et d'OpenSSL qui supportent SNI, et si le navigateur du client le supporte aussi, alors le nom d'hôte sera inclus dans la requête SSL originale, et le serveur web pourra sélectionner le bon serveur virtuel SSL.

Bien entendu, vous pouvez utiliser l'hébergement virtuel basé sur le nom pour identifier de nombreux hôtes virtuels non-SSL (tous sur le port 80 par exemple), et ne gérer qu'un seul hôte virtuel SSL (sur le port 443). Mais dans ce cas, vous devez définir le numéro de port non-SSL à l'aide de la directive NameVirtualHost dans ce style :

```
NameVirtualHost 192.168.1.1:80
```

il existe d'autres solutions alternatives comme :

Utiliser des adresses IP différentes pour chaque hôte SSL. Utiliser des numéros de port différents pour chaque hôte SSL.

Comment mettre en oeuvre la compression SSL ?

Bien que la négociation pour la compression SSL ait été définie dans la spécification de SSLv2 et TLS, ce n'est qu'en mai 2004 que la RFC 3749 a défini DEFLATE comme une méthode de compression standard négociable.

Depuis la version 0.9.8, OpenSSL supporte cette compression par défaut lorsqu'il est compilé avec l'option `zlib`. Si le client et le serveur supportent la compression, elle sera utilisée. Cependant, la plupart des clients essaient encore de se connecter avec un Hello SSLv2. Comme SSLv2 ne comportait pas de table des algorithmes de compression préférés dans sa négociation, la compression ne peut pas être négociée avec ces clients. Si le client désactive le support SSLv2, un Hello SSLv3 ou TLS peut être envoyé, selon la bibliothèque SSL utilisée, et la compression peut être mise en oeuvre. Vous pouvez vérifier si un client utilise la compression SSL en journalisant la variable `%{SSL_COMPRESS_METHOD}x`.

Lorsque j'utilise l'authentification de base sur HTTPS, l'icône de verrouillage des navigateurs Netscape reste ouverte quand la boîte de dialogue d'authentification apparaît. Cela signifie-t-il que les utilisateur et mot de passe sont envoyés en clair ?

Non, le couple utilisateur/mot de passe est transmis sous forme chiffrée. L'icône de chiffrement dans les navigateurs Netscape n'est pas vraiment synchronisé avec la couche SSL/TLS. Il ne passe à l'état verrouillé qu'au moment où la première partie des données relatives à la page web proprement dite sont transférées, ce qui peut prêter à confusion. Le dispositif d'authentification de base appartient à la couche HTTP, qui est située au dessus de la couche SSL/TLS dans HTTPS. Avant tout transfert de données HTTP sous HTTPS, la couche SSL/TLS a déjà achevé sa phase de négociation et basculé dans le mode de communication chiffrée. Ne vous laissez donc pas abuser par l'état de cet icône.

Pourquoi des erreurs d'entrée/sortie apparaissent-elles lorsqu'on se connecte via HTTPS à un serveur Apache+mod_ssl avec des versions anciennes de Microsoft Internet Explorer (MSIE) ?

La première raison en est la présence dans l'implémentation SSL de certaines versions de MSIE de bogues subtils en rapport avec le

dispositif de "maintien en vie" (keep-alive) HTTP, et les alertes de notification de fermeture de session SSL en cas de coupure de la connexion au point d'entrée (socket). De plus, l'interaction entre SSL et les fonctionnalités HTTP/1.1 pose problème avec certaines versions de MSIE. Vous pouvez contourner ces problèmes en interdisant à Apache l'utilisation de HTTP/1.1, les connexions avec maintien en vie ou l'envoi de messages de notification de fermeture de session SSL aux clients MSIE. Pour cela, vous pouvez utiliser la directive suivante dans votre section d'hôte virtuel avec support SSL :

```
SetEnvIf User-Agent "MSIE [2-5]" \
         nokeepalive ssl-unclean-shutdown \
         downgrade-1.0 force-response-1.0
```

En outre, certaines versions de MSIE ont des problèmes avec des algorithmes de chiffrement particuliers. Hélas, il n'est pas possible d'apporter une solution spécifique à MSIE pour ces problèmes, car les algorithmes de chiffrement sont utilisés dès la phase de négociation SSL. Ainsi, une directive `SetEnvIf` spécifique à MSIE ne peut être d'aucun secours. Par contre, vous devrez ajuster les paramètres généraux de manière drastique. Avant de vous décider, soyez sûr que vos clients rencontrent vraiment des problèmes. Dans la négative, n'effectuez pas ces ajustements car ils affecteront *tous* vos clients, ceux utilisant MSIE, mais aussi les autres.

Comment activer TLS-SRP ?

Le protocole TLS-SRP (Echange de clés sécurisé par mot de passe pour TLS comme spécifié dans la RFC 5054) peut compléter ou même remplacer les certificats lors du processus d'authentification des connexions SSL. Pour utiliser TLS-SRP, spécifiez un fichier de vérification SRP OpenSSL via la directive `SSLSRPVerifierFile`. Vous pouvez créer le fichier de vérification via l'utilitaire `openssl` :

```
openssl srp -srpvfile passwd.srpv -add username
```

Une fois ce fichier créé, vous devez le référencer dans la configuration du serveur SSL :

```
SSLSRPVerifierFile /path/to/passwd.srpv
```

Pour forcer les clients à utiliser des algorithmes de chiffrement basés sur TLS-SRP et s'affranchissant des certificats, utilisez la directive suivante :

```
SSLCipherSuite "!DSS:!aRSA:SRP"
```

Pourquoi des erreurs de négociation apparaissent avec les clients basés sur Java lorsqu'on utilise un certificat de plus de 1024 bits ?

Depuis la version 2.4.7, `mod_ssl` utilise des paramètres DH qui comportent des nombres premiers de plus de 1024 bits. Cependant, java 7 et ses versions antérieures ne supportent que les nombres premiers DH d'une longueur maximale de 1024 bits.

Si votre client basé sur Java s'arrête avec une exception telle que `java.lang.RuntimeException: Could not generate DH keypair` et `java.security.InvalidAlgorithmParameterException: Prime size must be multiple of 64, and can only range from 512 to 1024 (inclusive)`, et si httpd enregistre le message `tlsv1 alert internal error (SSL alert number 80)` dans son journal des erreurs (avec un `LogLevel info` ou supérieur), vous pouvez soit réarranger la liste d'algorithmes de mod_ssl via la directive `SSLCipherSuite` (éventuellement en conjonction avec la directive `SSLHonorCipherOrder`), soit utiliser des paramètres DH personnalisés avec un nombre premier de 1024 bits, paramètres qui seront toujours prioritaires par rapport à tout autre paramètre DH par défaut.

Pour générer des paramètres DH personnalisés, utilisez la commande `openssl dhparam 1024`. Vous pouvez aussi utiliser les paramètres DH standards issus de la RFC 2409 (↗ http://www.ietf.org/rfc/rfc2409.txt) , section 6.2 :

```
-----BEGIN DH PARAMETERS-----
MIGHAoGBAP//////////yQ/aoiFowjTExmKLgNwc0SkCTgiKZ8x0Agu+pjsTmyJR
Sgh5jjQE3e+VGbPNOkMbMCsKbfJfFDdP4TVtbVHCReSFtXZiXn7G9ExC6aY37WsL
/1y29Aa37e44a/taiZ+lrp8kEXxLH+ZJKGZR7OZTgf//////////AgEC
-----END DH PARAMETERS-----
```

Ajoute les paramètres personnalisés incluant les lignes "BEGIN DH PARAMETERS" et "END DH PARAMETERS" à la fin du premier fichier de certificat défini via la directive `SSLCertificateFile`.

Support de mod_ssl

- Quelles sont les sources d'informations disponibles en cas de problème avec mod_ssl ?
- Qui peut-on contacter pour un support en cas de problème avec mod_ssl ?
- Quelles informations dois-je fournir lors de l'écriture d'un rapport de bogue ?
- Un vidage mémoire s'est produit, pouvez-vous m'aider ?
- Comment puis-je obtenir une journalisation de ce qui s'est passé, pour m'aider à trouver la raison de ce vidage mémoire ?

Quelles sont les sources d'informations disponibles en cas de problème avec mod_ssl ?

Voici les sources d'informations disponibles ; vous devez chercher ici en cas de problème.

Vous trouverez des réponses dans la Foire Aux Questions du manuel utilisateur (ce document)
http://httpd.apache.org/docs/2.4/ssl/ssl_faq.html
Cherchez tout d'abord dans la foire aux questions (ce document). Si votre question est courante, on a déjà dû y répondre de nombreuses fois, et elle fait probablement partie de ce document.

Qui peut-on contacter pour un support en cas de problème avec mod_ssl ?

Voici toutes les possibilités de support pour mod_ssl, par ordre de préférence. Merci d'utiliser ces possibilités *dans cet ordre* - ne vous précipitez pas sur celle qui vous paraît la plus alléchante.

1. *Envoyez un rapport de problème à la liste de diffusion de support des utilisateurs d'Apache httpd*
   users@httpd.apache.org
   C'est la deuxième manière de soumettre votre rapport de problème. Ici aussi, vous devez d'abord vous abonner à la liste, mais vous pourrez ensuite discuter facilement de votre problème avec l'ensemble de la communauté d'utilisateurs d'Apache httpd.
2. *Ecrire un rapport de problème dans la base de données des bogues*
   http://httpd.apache.org/bug_report.html
   C'est la dernière manière de soumettre votre rapport de problème. Vous ne devez utiliser cette solution que si vous avez déjà écrit aux listes de diffusion, et n'avez pas trouvé de solution. Merci de suivre les instructions de la page mentionnée ci-dessus *avec soin*.

Quelles informations dois-je fournir lors de l'écriture d'un rapport de bogue ?

Vous devez toujours fournir au moins les informations suivantes :

Les versions d'Apache httpd et OpenSSL installées
La version d'Apache peut être déterminée en exécutant `httpd -v`. La version d'OpenSSL peut être déterminée en exécutant `openssl version`. Si Lynx est installé, vous pouvez aussi exécuter la commande`lynx -mime_header http://localhost/ | grep Server` et ainsi obtenir ces informations en une seule fois.

Les détails de votre installation d'Apache httpd et OpenSSL
A cet effet, vous pouvez fournir un fichier journal de votre session de terminal qui montre les étapes de la configuration et de l'installation. En cas d'impossibilité, vous devez au moins fournir la ligne de commande `configure` que vous avez utilisée.

En cas de vidage mémoire, inclure une trace de ce qui s'est passé
Si votre serveur Apache httpd fait un vidage de sa mémoire, merci de fournir en pièce jointe un fichier contenant une trace de la zone dédiée à la pile (voir ci-dessous pour des informations sur la manière de l'obtenir). Il est nécessaire de disposer de ces informations afin de pouvoir déterminer la raison de votre vidage mémoire.

Une description détaillée de votre problème
Ne riez pas, nous sommes sérieux ! De nombreux rapports n'incluent pas de description de la véritable nature du problème. Sans ces informations, il est très difficile pour quiconque de vous aider. Donc, et c'est votre propre intérêt (vous souhaitez que le problème soit résolu, n'est-ce pas ?), fournissez, s'il vous plait, le maximum de détails possible. Bien entendu, vous devez aussi inclure tout ce qui a été dit précédemment.

Un vidage mémoire s'est produit, pouvez-vous m'aider ?

En général non, du moins tant que vous n'aurez pas fourni plus de détails à propos de la localisation dans le code où Apache a effectué son vidage mémoire. Ce dont nous avons en général besoin pour vous aider est une trace de ce qui s'est passé (voir la question suivante). Sans cette information, il est pratiquement impossible de déterminer la nature du problème et de vous aider à le résoudre.

Comment puis-je obtenir une journalisation de ce qui s'est passé, pour m'aider à trouver la raison de ce vidage mémoire ?

Vous trouverez ci-dessous les différentes étapes permettant d'obtenir une journalisation des évènements (backtrace) :

1. Assurez-vous que les symboles de débogage sont disponibles, au moins pour Apache. Pour cela, sur les plates-formes où GCC/GDB est utilisé, vous devez compiler Apache+mod_ssl avec l'option ``OPTIM="-g -ggdb3"``. Sur les autres plates-formes, l'option ``OPTIM="-g"`` est un minimum.
2. Démarrez le serveur et essayez de reproduire le vidage mémoire. A cet effet, vous pouvez utiliser une directive du style ``CoreDumpDirectory /tmp" pour être sûr que le fichier de vidage mémoire puisse bien être écrit. Vous devriez obtenir un fichier `/tmp/core` ou `/tmp/httpd.core`. Si ce n'est pas le cas, essayez de lancer votre serveur sous un UID autre que root. Pour des raisons de sécurité, de nombreux noyaux modernes de permettent pas à un processus de vider sa mémoire une fois qu'il a accompli un `setuid()` (à moins qu'il effectue un `exec()`) car des informations d'un niveau privilégié pourraient être transmises en mémoire. Si nécessaire, vous pouvez exécuter `/chemin/vers/httpd -X` manuellement afin de ne pas permettre à Apache de se clôner (fork).
3. Analysez le vidage mémoire. Pour cela, exécutez `gdb /path/to/httpd /tmp/httpd.core` ou une commande similaire. Dans GDB, tout ce que vous avez à faire est d'entrer `bt`, et voila, vous obtenez la backtrace. Pour les débogueurs autres que GDB consulter le manuel correspondant.

Commentaires

Notice:
This is not a Q&A section. Comments placed here should be pointed towards suggestions on improving the documentation or server, and may be removed again by our moderators if they are either implemented or considered invalid/off-topic. Questions on how to manage the Apache HTTP Server should be directed at either our IRC channel, #httpd, on Freenode, or sent to our mailing lists.