SQL Net – Implémentation du cryptage SSL/TLS avec certificats

Cyril Wasmer, dbi-services

Oracle offre la possibilité de crypter la communication sur la couche SQL Net depuis de nombreuses années et avec plusieurs variantes disponibles. La grande « nouveauté » (depuis 2014) réside dans le fait que maintenant cette option, anciennement partie de l'option payante « Advanced Security » (ASO), est disponible gratuitement dans toutes les versions supportées des bases de données Oracle (cf. Database Licensing Information guide).

Pour rappel, les différentes options de cryptage disponibles pour la couche SQL Net sont :

- Encryptage ASO de base, avec le choix des protocoles DES, 3DES, RC4 et AES (via les paramètres sqlnet.encryption_*). Vous pouvez consulter à ce propos l'article de Yann Neuhaus dans la newsletter du SOUG 4/2014.
- Kerberos ou Radius avec des serveurs/tokens d'authentification dédiés.
- SSL/TLS, qui va faire l'objet de cet article.

Contexte

SSL/TLS étant basée sur de la cryptographie à clé publique, sa mise en place nécessite la création et la gestion de certificats. L'avantage de ce type de cryptage est qu'il permet d'une part de ne pas avoir besoin d'échanger de clé entre l'émetteur et le récepteur, mais qu'il est aussi possible de l'utiliser à des fins d'authentification des parties prenantes.

Le principe de base se repose sur la génération de paires de clés privées/publiques, avec communication de la clé publique uniquement.

Un certificat (norme X.509) est simplement une « carte d'identité »,

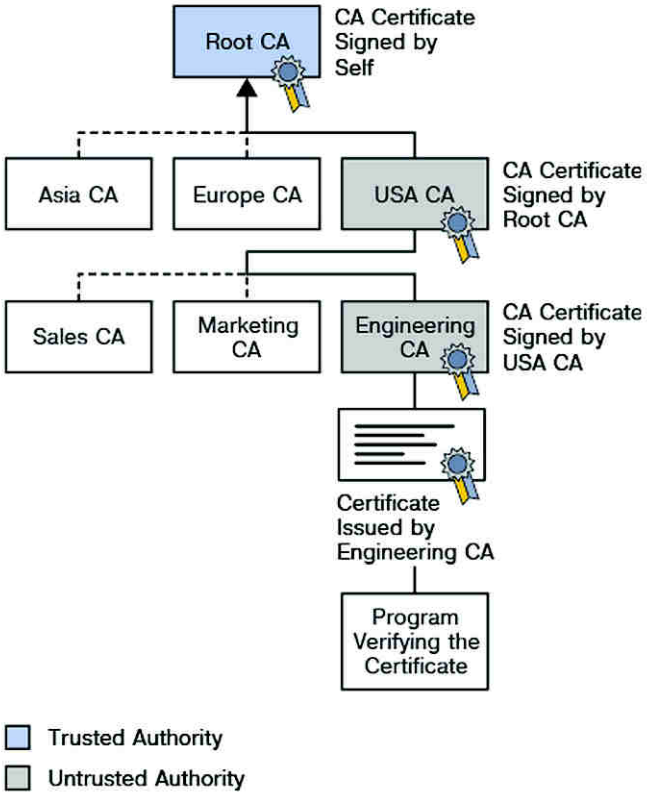

Figure 1

```
$ mkdir /u01/app/oracle/admin/wallet
$ chmod 700 /u01/app/oracle/admin/wallet
$ cd /u01/app/oracle/admin/wallet
$ orapki wallet create -wallet wallet-mydbserver -auto_login
```

Listage 1

signée par la clé privée d'une « autorité de confiance ». On y retrouve des informations déclaratives du type :

- Dates de validités
- Identification ou « Distinguished Name » (DN), composé de divers champs : C, ST, L, O, OU, CN dont nous parlerons plus tard ; numéro de série, adresse mail
- Type de certificat

Afin de vérifier l'authenticité du certificat, il suffit de le décrypter avec la clé publique de l'autorité de confiance. Si le décryptage avec la clé publique fonctionne, alors il a bien été crypté avec la clé privée !

Toute la problématique de la cryptographie à clé publique réside donc dans le fait d'avoir « confiance » en l'autorité de... confiance.

C'est pourquoi il a été créé, dans cette infrastructure (aussi appelée « Public Key Infrastructure » ou PKI), des entités spécifiques étant réputées de confiance et nommées « autorités de certificat » (« Certificate Authority » ou CA).

Ces entités ont pour but de signer, avec leur clé privée, les demandes de certificats qui leur sont envoyées (après vérification de la validité de la demande).

Les certificats X.509 de ces entités (appelés spécifiquement « certificats racines ») sont connus par tous, et sont même enregistrés par défaut dans les systèmes d'exploitation, navigateur web ou applications des postes utilisateurs. Ce sont eux qui permettent de vérifier la validité des certificats des sites internet configurés en https par exemple.

D'un point de vue organisationnel, il est possible d'utiliser non pas juste une autorité de certification, mais une hiérarchie fonctionnelle *(Figure 1)*.

Dans ce cas de figure, la validation de la chaîne de certificat est effectuée en remontant :

- Certificat client signé par certificat « Engineering CA »
- Certificat « Engineering CA » signé par certificat « USA CA »
- Certificat « USA CA » signé par « Root CA »
  - Si le certificat « Root CA » est dans la liste des certificats de confiance, le certificat signé « USA CA » est bien valide
  - Le certificat « Engineering USA », signé par « USA CA », est lui aussi valide
  - Par conséquent, le certificat du client est valide

Oracle utilise des fichiers nommés « wallet » (norme PKCS#12) afin de stocker toutes les informations nécessaires au fonctionnement de SSL. Ces

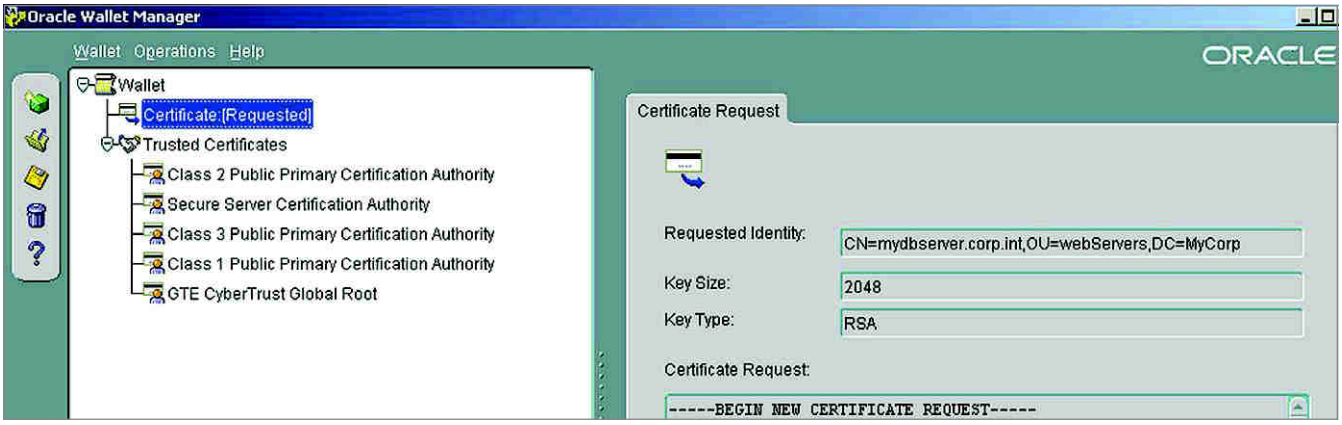

Figure 2

fichiers peuvent contenir des informations d'authentification mais aussi des certificats, clés privées... Ces wallets sont utilisés non seulement dans le cadre des certificats SSL, mais aussi pour l'option Transparent Data Encryption, voire pour y stoker des informations d'authentification de bases de données (voir à ce sujet le blog de Nicolas Jardot : Use a wallet to encrypt Oracle client passwords)

Afin de configurer le cryptage SSL/TLS dans SQL Net, il va donc nous falloir générer un wallet valide et appliquer le cryptage voulu dans la couche SQL Net. Ce wallet va contenir, en plus du certificat signé par notre CA, la demande envoyée au CA, ou « Certificate Signing Request » CSR, mais aussi l'ensemble de la chaîne des certificats nécessaires à sa validation.

Génération du Wallet

Dans notre projet, le client souhaite protéger les accès à la base de données pour toutes les opérations d'administration ou de changement de mot de passe, sans impacter les accès clients existants. Il souhaite une authentification au niveau du serveur, les clients n'auront pas besoin de s'authentifier (et donc n'auront pas besoin de certificats de leur côté).

Oracle offre deux possibilités afin de gérer les wallets, soit en ligne de commande via « orapki », ou en interface graphique, via « Oracle Wallet Manager », ou « owm ». Nous allons autant que possible privilégier la ligne de commande.

La première étape afin de générer notre wallet est de le créer dans un nouveau répertoire *(Listage 1)*.

La dernière commande va vous demander d'entrer un mot de passe et va ensuite générer un fichier « ewallet.p12 » dans le nouveau sous-répertoire « ./wallet-mydbserver ».

Le paramètre « –auto-login » permettra de pouvoir utiliser ce wallet pour les opérations PKI sans avoir à saisir son mot de passe, il va pour cela générer un fichier « cwallet.sso » dans le même répertoire. Vous aurez toutefois toujours besoin du mot de passe lors des manipulations sur le fichier wallet. Il est primordial de gérer avec parcimonie l'accès à ce répertoire et ces fichiers.

Oracle recommande de protéger aussi ces fichiers contre toute suppression importune (via les commandes « chattr » ou « chmod » sous Linux) et d'en faire des sauvegardes fréquentes.

Notre fichier wallet créé, nous allons y ajouter le certificat de notre serveur

```
$ orapki wallet add -wallet wallet-mydbserver -dn
"cn=mydbserver.corp.int, ou=webServers, dc=MyCorp" -keysize 2048
```

Listage 2

```
$ orapki wallet export -wallet wallet-mydbserver -dn
"cn=mydbserver.corp.int, ou=webServers, dc=MyCorp" -request
mydbserver.csr
```

Listage 3

```
$ openssl ca -policy policy_anything -in mydbserver.csr -out
mydbserver.cer
```

Listage 4

```
$ orapki wallet add -wallet wallet-mydbserver -trusted_cert
-cert MyCorp_RootCertificate.cer
$ orapki wallet add -wallet wallet-mydbserver -trusted_cert
-cert MyCorp_GoldCertificate.cer
```

Listage 5

« mydbserver ». Ce certificat est simplement un ensemble d'information permettant de définir, sans ambiguïté, l'objet du certificat. Pour cela des champs prédéfinis sont à remplir :

- CN / « CommonName » : nom de l'entité à certifier (ici son FQN: « mydbserver.corp.int »)
- OU / « Organization Unit » : peut correspondre au département pour lequel le serveur est utilisé (ici: « webServers »)
- DC / « DomainComponent » : le nom de l'entreprise (ici: « MyCorp »)

Toutes ces informations vont nous générer le DN, ou « Distinguished Name », de notre entité.

Il est à noter que bien d'autres champs sont disponibles pour constituer le DN, et selon les applications certains peuvent même être obligatoires.

Nous ajoutons notre DN dans notre wallet, en précisant une taille de clé RSA de 2048bits *(Listage 2)*.

Afin d'avoir un apperçu du contenu de notre Wallet, nous pouvons lancer l'application « Oracle Wallet Manager » et ouvrir notre Wallet *(Figure 2)*.

Nous voyons notre certificat en mode « Requested » ainsi que d'autres, classés dans la catégorie des « Trusted Certificate ». Ces derniers correspondent à des Authoritées de Certification publiques, comme Verisign ou Cybertrust. Elles sont nécessaires dans le cas de certificats signés par ces entitées, mais pas dans notre cas où le Root Certificate de l'entreprise Mycorp est lui-même auto-signé. Nous pouvons donc supprimer tous ces certificats pour gagner en lisibilité.

Nous allons maintenant exporter une demande de certificat (CSR) afin de la faire signer par un certificat de notre entreprise « MyCorp » *(Listage 3)*.

Ce qui va nous générer le fichier « mydbserver.csr ».

Ce fichier sera à envoyer aux services PKI ou sécurité de l'entreprise afin de le faire signer par le certificat racine (ou un autre certificat valide de l'arborescence PKI). Cette opération nécessite d'avoir la clé privée du certificat ainsi que son mot de passe *(Listage 4)*.

Avec ce fichier de certificat signé par notre certificat racine, nous allons pouvoir complèter notre wallet avec les certificats nécessaires à sa validation.

Chez notre client MyCorp, nous avons une hiérarchie PKI à deux niveaux, où le certificat racine « RootCertificate » a signé un « Gold Certificate » qui lui-même a signé le certificat de notre serveur.

Nous avons donc besoin d'importer comme « Trusted Certificates » ces deux certificats (dans l'ordre) *(Listage 5)*.

Il nous faut aussi importer le « User Certificate » de notre serveur, qui nous a été signé par le « Gold Certificate » *(Listage 6)*.

Afin de s'assurer que notre certificat est bien valide, vous pouvez ouvrir votre Wallet avec « Oracle Wallet Manager ». Son status doit être « Ready » *(Figure 3)*.

Si vous obtenez une erreur PKI-04004 ou PKI-04006 lors de l'ajout du certificat utilisateur via orapki, assurez-vous que le wallet dans lequel vous essayez de l'importer est bien celui qui a servi pour exporter la demande .csr, et que le certificat est bien en status « Requested ». Vérifiez aussi que vos « Trusted Certificate » sont tous présents.

Figure 3

```
$ orapki wallet add -wallet wallet-mydbserver -user_cert -cert
mydbserver.cer
```

Listage 6

```
WALLET_LOCATION =
   (SOURCE =
       (METHOD = FILE)
       (METHOD_DATA =
           (DIRECTORY = /u01/app/oracle/admin/wallet/)
       )
   )
```

Listage 7

```
SSL_CLIENT_AUTHENTICATION = FALSE
SSL_SERVER_AUTHENTICATION = TRUE
```

Listage 8

```
SSL_CIPHER_SUITES=(SSL_RSA_WITH_AES_128_CBC_SHA, SSL_RSA_WITH_
AES_256_CBC_SHA)
```

Listage 9

```
SSL_VERSION = 1.2
```

Listage 10

```
LISTENER_SSL =
   (DESCRIPTION_LIST =
       (DESCRIPTION =
           (ADDRESS =
               (PROTOCOL = TCPS)
               (Host = mydbserver.corp.int)
               (Port = 2484)
           )
       )
   )
SID_LIST_LISTENER_SSL=
   (SID_LIST=
       (SID_DESC =
           (GLOBAL_DBNAME = mydb.corp.int)
           (SID_NAME      = mydb)
           (ORACLE_HOME   = /u01/app/oracle/product/11.2.0.4/db_1)
       )
   )
```

Listage 11

```
SSL_VERSION = 1.2
SSL_CIPHER_SUITES=(SSL_RSA_WITH_AES_128_CBC_SHA,SSL_RSA_WITH_
AES_256_CBC_SHA)
SSL_CLIENT_AUTHENTICATION = FALSE
SSL_SERVER_AUTHENTICATION = TRUE
WALLET_LOCATION =
   (SOURCE =
       (METHOD = FILE)
       (METHOD_DATA =
           (DIRECTORY = /u01/app/oracle/admin/wallet/)
       )
   )
```

Listage 12

```
$ openssl s_client -connect mydbserver.corp.int:2484

CONNECTED(00000004)
depth=2 DC = MyCorp, OU = MyCorp Internal Root CA
verify error:num=19:self signed certificate in certificate chain
verify return:0
---
Certificate chain
 0 s:/DC=MyCorp/OU=webServers/CN=mydbserver.corp.int
   i:/DC=MyCorp/OU=CAs/OU=MyCorp Internal Gold CA1
 1 s:/DC=MyCorp/OU=CAs/OU=MyCorp Internal Gold CA1
   i:/DC=MyCorp/OU=CAs/OU=MyCorp Internal Root CA
 2 s:/DC=MyCorp/OU=CAs/OU=MyCorp Internal Root CA
   i:/DC=MyCorp/OU=CAs/OU=MyCorp Internal Root CA
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIEWz……Usapie
-----END CERTIFICATE-----
```

Listage 13

Si c'est bien le cas, utilisez Oracle Wallet Manager pour importer le certificat utilisateur sans provoquer ces erreurs.

Modification SQL Net

Maintenant que nous avons un wallet valide, nous allons configurer le listener en TCPS avec les paramètres de cryptage souhaités. Dans notre exemple, la connexion en SSL/TLS ne sera nécessaire que pour les opérations d'administration ou de changement de mot de passe, nous allons donc ajouter un nouveau listener.

Plusieurs paramètres du fichier listener.ora vont impacter notre cryptage. Tout d'abord, nous devons indiquer où se trouve notre fichier wallet *(Listage 7)*.

Nous allons ensuite définir les modes d'authentification des clients et du serveur. Dans notre exemple, nous souhaitons que le serveur soit authentifié auprès des clients, mais pas l'inverse. Cela nous permet de ne gérer qu'un certificat au niveau du serveur; dans le cas contraire nous devrions générer un certificat par client.

Nous positionnons donc les valeurs suivantes dans notre fichier listener.ora *(Listage 8)*.

Vient ensuite le choix des « suites cryptographiques ». Ces dernières sont un ensemble de protocoles de cryptage, d'authentification et d'intégrité de données.

Nous souhaitons, dans notre exemple, une authentification faite par RSA, que l'intégrité des données soit calculée par SHA-1 et que le cryptage soit par AES en mode CBC de niveau 128 ou 256 bits *(Listage 9)*.

Enfin nous pouvons définir la version SSL/TLS minimale requise pour se connecter au serveur. Suite aux différentes attaques qui ont touchés les versions SSL/TLS, il est recommandé de ne plus utiliser SSL v3 mais TLS (ici en version 1.2) *(Listage 10)*.

Nous allons maintenant ajouter un nouveau listener TCPS sur le port 2484, avec une entrée statique pour notre base *(Listage 11)*.

Enfin nous ajoutons les paramètres SSL à notre fichier sqlnet.ora *(Listage 12)*.

Après un redémarrage du listener pour prise en compte de ces nouveaux paramètres, il ne nous reste plus qu'à créer une nouvelle entrée dans le fichier tnsnames.ora, configurée en protocole TCPS et sur le port 2484.

Pour s'assurer du fonctionnement de la couche SSL de notre nouveau listener, la solution la plus efficace consiste à effectuer une tentative d'authentification grâce à l'application open_ssl *(Listage 13)*.

L'erreur 19 indiquée est due au fait que, dans notre exemple, le certificat racine est un certificat auto-signé, nous pouvons l'ignorer.

Nous pouvons aussi comparer avec une capture tcpdump les informations transmises lors d'une connexion. Sans cryptage, nous pouvons voir toutes les commandes SQL ainsi que leurs résultats. Avec le cryptage activé, nous pouvons voir le « handshake » SSL avec le partage des certificats et une fois la connexion établie, plus que des paquets cryptés.

Conclusion

Maintenant que les options de cryptage SSL ne sont plus payantes dans les dernières versions d'Oracle, étant donné sa relative facilité d'implémentation et au vu du faible surcout en terme d'utilisation CPU et de bande passante, il nous est désormais très facile de proposer cette option à nos clients, et, si disponible, à bénéficier de son infrastructure PKI existante.

Dans notre projet, nous avons créé un nouveau listener pour accèder d'une manière totalement sécurisée à notre base de données afin d'effectuer les opérations d'administration, de changements de mot de passe… Il est aussi tout à fait envisageable de mettre en place le cryptage pour les accès clients, mais une étude de performance est recommandée avant mise en production.

Une fois cette option en place, il sera non seulement impossible de décrypter le traffic entre la base de donnée et ses clients, mais elle permettra aussi d'empêcher les attaques de type « Man In The Middle » grâce à l'échange des certificats.

Sources

- Database Security Guide – 12.1 – Part IV + Part V
- Oracle Database Security Guide – 12.1 – Configuring Secure Sockets Layer Authentication
- Database Net Services Reference – 12.1 – All SSL_* parameters

Cyril Wasmer