MÉNARD / LOUINEAU / MIE
Lucas / Noah / Martin

Sécurisation des,connexion vers un serveur MariaDB

sudo openssl req -new -x509 -nodes -days 365000 -key ca-key.pem -out ca-cert.pem
openssl genrsa 2048 > ca-key.pem

Création d'un utilisateur avec tous les droit d'administration afin de se connecter au serveur mariadb à distance :

Faire ses commandes dans le serveur pas dans mariadb :

Modifier l'option Bind dans le fichier:

sudo nano /etc/mysql/mariadb.conf.d/50-server.cnf

```
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
bind-address            = 0.0.0.0
```

L'option bind-address permet d'autoriser les connexions depuis une adresse IP. En mettant l'adresse 0.0.0.0 cela permet d'autoriser les connexions depuis n'importe quelles adresses.

Redémarrer mariadb avec la commande : sudo systemctl restart mariadb-server

---

Dans Mariadb :

GRANT ALL PRIVILEGES ON *.* TO 'dba'@'%'IDENTIFIED BY 'password' WITH GRANT OPTION;

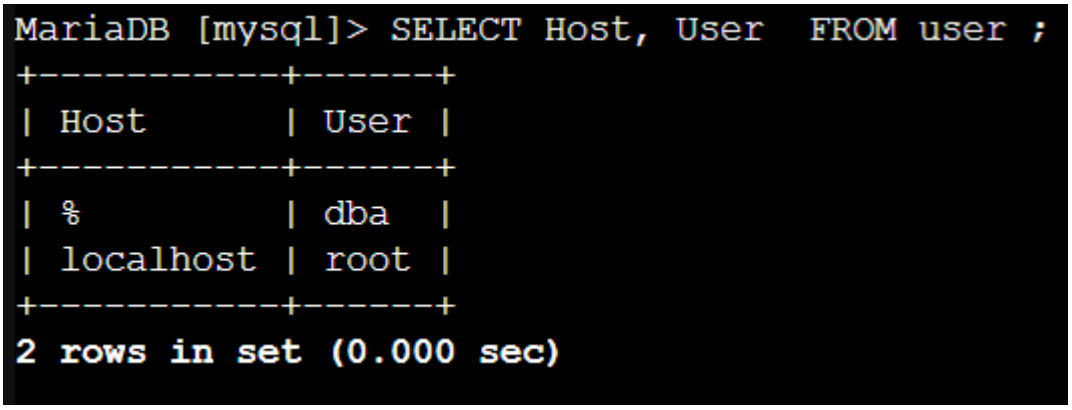
Cette commande permet de créer un utilisateur dba avec pour mot de passe "password" avec tous les privilèges sur le réseau "%" donc tous les réseaux et toutes les options.

Afin de vérifier que la création de l'utilisateur s'est bien effectuée on va vérifier en affichant tous les utilisateurs avec les commandes suivantes :

USE mysql ;

il faut se connecter à la base de donnée mysql afin de pouvoir afficher les utilisateurs présents.

Puis tapper la commande :
SELECT Host, User FROM user ;



Et enfin tapper la commande FLUSH PRIVILEGES; afin d'appliquer les changements

Pour se connecter à distance à cette utilisateur via le terminal windows :

mysql -u dba -p -h (172.16.254.131)→ à modifier en fonction de son serveur

---

Ordre des tâches :

1- Créer un certificat CA → ca-cert.pem
2- Créer un certificat serveur → server-req.pem
3- Signe le certificat serve avec le fichier ca-cert.pem
4- Créer le certificat client
5- Signer le certificat client avec le fichier ca-cert.pem → client-cert.pem
6- Tester les certificats

SSL :

Générer un certificat SSL et une clé privée :

a. Installez OpenSSL : sudo apt-get install openssl

b. Générez un certificat SSL et une clé privée :
sudo openssl req -newkey rsa:2048 -nodes -keyout /etc/mysql/ssl/server-key.pem -out /etc/mysql/ssl/server-cert.pem
Cette commande génère une clé privée dans /etc/mysql/ssl/server-key.pem et un certificat SSL auto-signé dans /etc/mysql/ssl/server-cert.pem

c. Ouvrez le fichier de configuration de MariaDB:
/etc/mysql/mariadb.conf.d/50-server.cnf

Ajoutez les paramètres de configuration suivants à la section [mysqld] du fichier de configuration :
ssl-ca=/etc/mysql/ssl/ca-cert.pem
ssl-cert=/etc/mysql/ssl/server-cert.pem
ssl-key=/etc/mysql/ssl/server-key.pem
sudo openssl req -newkey rsa:2048 -nodes -keyout /etc/mysql/ssl/server-key.pem -out /etc/mysql/ssl/server-cert.pem
openssl genpkey -out ca-cert.key -algorithm RSA -pkeyopt rsa_keygen_bits:2048

openssl rsa -pubout -in ca-cert.key -out ca-cert.pem

sudo openssl x509 -req -in /etc/mysql/ssl/server-key.pem -CA /etc/mysql/ssl/ca-cert.pem -CAkey /etc/mysql/ssl/ca-key.pem -CAcreateserial -out /etc/mysql/ssl/server-cert.pem -days 365000

sudo openssl req -newkey rsa:2048 -nodes -keyout /etc/mysql/ssl/client-key.pem -out /etc/mysql/ssl/client-cert.pem

sudo openssl x509 -req -in /etc/mysql/ssl/client-key.pem -CA /etc/mysql/ssl/ca-cert.pem -CAkey /etc/mysql/ssl/ca-key.pem -CAcreateserial -out /etc/mysql/ssl/server-cert.pem -days 365000

test :

mysql -u dba -p --ssl-ca=/etc/mysql/ssl/ca-cert.pem --ssl-cert=/etc/mysql/ssl/client-cert.pem --ssl-key=/etc/mysql/ssl/client-key.pem

X.509

Pourquoi “standard X.509” et à quoi ça sert de l'utiliser ici ?

Dans ce contexte, le mot “standard” établit un ensemble de règles et de normes communes pour la création, la validation et l'utilisation de certificats numériques, de manière à ce que ces certificats soient compatibles entre différents systèmes et différents utilisateurs. On utilise X.509 car il est pris en compte par MariaDB. Utilise le protocole PKCS 10

Pourquoi un certificat auto-signé ?

Un certificat auto-signé c’est un certificat qu’un utilisateur émet en son propre nom via une création de celle-ci.

Création de certificat auto-signé :

sources :
https://learn.microsoft.com/fr-fr/azure/iot-hub/reference-x509-certificates#self-signed-certificates

+ chatGPT pour les détails de commande

En utilisant OPENSSL : (plus d’information sur OPENSSL dans la partie SSL/TLS du compte-rendu)

Je peux donc créer un certificat auto-signé à l'aide de la commande ci-dessous.

openssl genpkey -out ca-cert.key -algorithm RSA -pkeyopt rsa_keygen_bits:2048

Détails de la commande :

openssl : C'est le nom de l'outil en ligne de commande qui est utilisé pour générer la clé privée.

genpkey : C'est l'option utilisée pour générer une clé privée.

-out {KeyFile} : C'est l'option qui spécifie le nom et le chemin du fichier de sortie où la clé privée sera enregistrée. {KeyFile} doit être remplacé par le nom que vous souhaitez donner au fichier.

-algorithm RSA : C'est l'option qui spécifie l'algorithme de chiffrement à utiliser pour la génération de la clé privée. Dans ce cas, RSA est utilisé, ce qui signifie que la clé sera basée sur l'algorithme de chiffrement RSA.

-pkeyopt rsa_keygen_bits:2048 : C'est l'option qui spécifie la longueur de la clé privée en bits. Dans ce cas, la clé privée aura une longueur de 2048 bits. Le nombre de bits est important car il détermine la force de la clé privée. Plus la clé est longue, plus elle est difficile à casser par force brute.

ci dessous on peut voir un exemple de génération de clé de certificat dans le fichier par défaut :

```
menardl@server--menardl:~$ openssl genpkey -out ca2.key -algorithm RSA -pkeyopt rsa_keygen_bits:2048
.............+++++
.....................................+++++
menardl@server--menardl:~$ ls
ca.key  ca2.key
menardl@server--menardl:~$ cat ca2.key
-----BEGIN PRIVATE KEY-----
MIIEvwIBADANBgkqhkiG9w0BAQEFAASCBKkwggSlAgEAAoIBAQDkIBAvZTeAlH80
z80Wa4gtFl7uTbjH2ONgTw76zlzTzWH+us4snOVCvtGPU+ZiD2LHIpr3eIdfuJYC
DAcdwojEfuZtchlfeQcPzyl4ws6YSLN0DMHVv9ymA/7APBUTRxl20dWzfLdwkETo
xD4EWl5FFMvt/QW1TI3i5AsNosdNYFITcqk2UxyMzsjkVutQRy2Q3BlNPLRuhowH
4icCr9sZHYR8hDE3jAskWfR6jqQKV8ieouzEoJ+a+tRfRR5LMJnOHJwUMjd5xG3r
yguISX5sCrkiz4DoAeLckaOET/VXAEOmuAxjE+0m8TvRIeZgkJLITG7pn0gZKp/Y
5mI0lB9fAgMBAAECggEBAI4RYcMCd/RlYesTlpcllJKvseMayA69haz+z0dmqOzG
pVasI6Yo96X/jCaEi59aEZwOcw/XIgd8vv77WpswAVjN5tcWDbMlfj5cOedQ+rRf
4lJNd5GUcOd66XXydPWSfkj3GO00qFETp7aGMFEprje7Yd0OjDWCPYKlnjSZwSDv
xG0ZyTGR0ALNnJllnDvgej2pElhGVDob+SKutiNV7Ar/pBcW4lBs08MH8ObQsHhX
VaOgwaI0L/RKlW8dfTaBFtd9ohEk36n68qjDdLIOhfYv4y5Z50vobeKHLaB9xnGv
HjPOSIFiv6+FhGPwuIDtaWr3DlrCr3T164i0MKEoAMECgYEA8psT/O97YX1758JY
mdCAF/CnSiY2+FjPfvEjERRjjBzYCQootKni2sJkESVjSTKAOrn7UMEopqUILrxk
Bqi3WcEmlPHF8yz4rExStrXzaYqjlYNr7SlsxzMu48ku2qjat0gbILckMddVPitG
AkW+c8K34PFqE48tnBTlE0NPjDECgYEA8LhSShsXzJfwJMRlWZsyKllLnlfhPS4D
TXoNJXLITe0IFOBkvFj6fXiHwiWteR6e2i8hVd657n7VF+hvFXodijYaz86xjwk5
64YpqGj9moPOxma+JvE3ZnwhWjVrQAq0TtHFOY0hnYP9D/dxW8iXOLk67UXdmqe8
Wyxdx5bN0I8CgYEAOBYbqtNuLEyJPlv/ys5/UAlmM79bXuAt+V+zCC9IwyDqW7aO
+4bLwX8CfYJoCpx6R+TFnppOl8CtqAiGcv+7ZAy7nvlRICbulDpSp8ygn2H+tXsn
0yMnLiTEjzQ8L+is0I/UVNdBAE8GGnDbd4+Cr5/QP+xRY7lnDX0OjtspG8ECgYA/
+/6DjCgLTeebHkkCRGpVL18WO623IV/nXCXaamQrsnGPewXLuqr6ZutPA8zUgPsg
a56kwIsThcsGi9XbE5Z+glsmGX7AOmb+3657AUlHoPl5ax3oI/3h0+TZoZFceHvs
53lRCt6Ai/DcKZ2MFAl+Hl43a3a4ARXj9isHPmJ+qQKBgQCIFN2EUxn+PxG7T5aB
rKP7VjcGmSGT/QLUnBksmadZDlJbobylv7QKUgY2tI+XbnrBBlU29G/iEu8HKDOz
B/DXCjoMuGJMMK8WKNaXqnyYg0gnyHqrUmlUHzcPKxInuqPeWGCxWEUFVCzcr54O
EtCBNTgUUHnCr4UBwIUvhxb66A==
-----END PRIVATE KEY-----
menardl@server--menardl:~$
```

Maintenant ont génère la clé publique à partir de la clé privée créée auparavant

Pour faire cela on utilise la commande suivante :

openssl rsa -pubout -in ca-cert.key -out ca-cert.pem

```
menardl@server--menardl:~$ openssl rsa -pubout -in ca2.key -out public_key.pem
writing RSA key
menardl@server--menardl:~$ ls
ca.key  ca2.key  public_key.pem
menardl@server--menardl:~$ cat public_key.pem
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA5CAQL2U3gNR/NM/NFmuI
LRZe7k24x9jjYE8O+s5c08lh/rrOLJzlQr7RjlPmYg9ixyKa93iHX7iWAgwHHcKI
xH7mbXIZX3kHD88teMLOmEizdAzBlb/cpgP+wDwVE0cddtHVs3y3cJBE6MQ+BFpe
RRTL7f0FtUyN4uQLDaLHTWBSE3KpNlMcjM7I5FbrUEctkNwdTTy0boaMB+InAq/b
GR2EfIQxN4wLJFn0eo6kClfInqLsxKCfmvrUX0UeSzCZzhycFDI3ecRt68oLiEl+
bAq5Is+A6AHi3JGjhE/lVwBDprgMYxPtJvE70SHmYJCSyExu6Z9IGSqf2OZiNJQf
XwIDAQAB
-----END PUBLIC KEY-----
menardl@server--menardl:~$
```

Détails de la commande :

-pubout : C'est l'option qui spécifie que la clé publique doit être générée à partir de la clé privée.

-in private_key.pem : C'est l'option qui spécifie le nom et le chemin de la clé privée RSA à partir de laquelle la clé publique doit être générée. Ici, la clé privée est stockée dans un fichier nommé "private_key.pem".

-out public_key.pem : C'est l'option qui spécifie le nom et le chemin du fichier où la clé publique doit être enregistrée. Ici, la clé publique sera stockée dans un fichier nommé "public_key.pem".

ci dessous on peut voir un exemple de génération de clé de certificat dans le fichier par défaut :

Le fichier "server-cert.pem" contient une clé publique qui permet de vérifier l'identité du serveur auprès du client, ainsi que des informations sur l'autorité de certification qui a émis le certificat. Il peut être utilisé en conjonction avec une clé privée pour établir une connexion sécurisée à l'aide du protocole SSL/TLS.

Comment les générer ? : **“openssl rsa -pubout -in server-key.pem -out server-cert.pem”**

Le fichier "server-key.pem" contient la clé privée associée au certificat du serveur MySQL. Cette clé est utilisée pour chiffrer et déchiffrer les données transmises entre le client et le serveur MySQL. Il est important de garder cette clé privée sécurisée et confidentielle, car toute personne qui y a accès peut potentiellement déchiffrer les données de communication SSL/TLS.

Comment les générer ? : **openssl genpkey -out server-key.pem -algorithm RSA -pkeyopt rsa_keygen_bits:2048**

Le fichier "ca.pem" est un fichier contenant le certificat de l'autorité de certification (CA). Une autorité de certification est une entité de confiance qui émet des certificats numériques pour des entités, telles que des serveurs Web ou des clients, pour garantir leur identité.

Comment les générer ? : ***openssl genpkey -out ca2.key -algorithm RSA -pkeyopt rsa_keygen_bits:2048 (commande déjà utilisé précédemment)***

Une fois que la clé privée est configurée et utilisée par le serveur, les clients peuvent établir des connexions sécurisées avec le serveur en utilisant le protocole SSL/TLS.

Par exemple, lorsqu'un utilisateur accède à un site web sécurisé, le navigateur du client établit une connexion SSL/TLS avec le serveur en utilisant la clé publique correspondant à la clé privée dans le fichier "server-key.pem". La connexion sécurisée permet de crypter les données échangées entre le client et le serveur, et de garantir l'authenticité et l'intégrité des données.

Certificat bidirectionnelle ou unidirectionnelle :

https://www.ibm.com/docs/fr/sig-and-i/10.0.0?topic=communication-one-way-two-way-ssl-authentication

Un certificat X.509 peut être utilisé pour une communication bidirectionnelle ou unidirectionnelle, tout dépend de la façon dont il est configuré.

Dans une communication unidirectionnelle, le certificat est utilisé **uniquement pour authentifier le serveur auprès du client**. Le client vérifie le certificat présenté par le serveur pour s'assurer qu'il s'agit bien du serveur qu'il souhaite contacter, et que la communication est sécurisée avec un chiffrement adéquat. Dans ce cas, le **certificat est utilisé uniquement pour la vérification de l'identité du serveur et le chiffrement des données envoyées du serveur au client.**

En revanche, dans une communication bidirectionnelle, le certificat est **utilisé pour l'authentification à la fois du serveur et du client.** Cela signifie que le client et le serveur doivent tous deux présenter des certificats pour vérifier leur identité respective. Dans ce cas, le **certificat est utilisé pour l'authentification mutuelle et le chiffrement des données échangées entre le client et le serveur.**

l'utilisation d'un certificat X.509 bidirectionnel est plus sécurisée que celle d'un certificat unidirectionnel, car elle permet d'authentifier les deux parties impliquées dans la communication. Cependant, cela peut nécessiter une configuration supplémentaire et une infrastructure de clé publique (PKI) plus complexe pour gérer les certificats des deux parties.

La PKI(Public Key Infrastructure) correspond aux politiques, aux rôles et aux procédures nécessaires pour créer, gérer, distribuer, utiliser, stocker et révoquer des certificats numériques.

Client ←→ Serveur

https://mariadb.com/kb/en/securing-connections-for-client-and-server/

Par défaut MariaDB transmet les données entre le serveur et le client sans les chiffrer. Si le client et le serveur s'exécutent dans le même hôte, cela ne pose pas de soucis car la sécurité est faite par d'autres moyens.
Dans le cas où le serveur et le client sont sur des réseaux séparés ou se trouvent dans un réseau à haut risque (ex: réseau bancaire, réseau de défense nationale, de santé ou encore télécommunication), le manque de cryptage peut introduire des problèmes de sécurités (ex : acteur extérieur malveillant peut écouter le trafic).

MariaDB permet de chiffrer les données en transit entre le serveur et les clients à l'aide du protocole TLS (SSL avant car ce dernier est maintenant considéré comme non sécurisé). La documentation utilise encore souvent le terme SSL et pour des raison de compatibilité, le système serveur lié à TLS et les variables d'état utilisent toujours le préfixe ssl mais MariaDB en interne ne prend en charge que ses successeurs sécurisés (comme TLS par exemple).

Pour sécuriser les connexions entre le serveur et le client il faut s'assurer que le serveur a été compilé avec le support TLS. Il faut également un certificat x509, une clé privée et la chaîne d'autorité de certification afin de vérifier le certificat x509 du serveur.

Pour que MariaDB utilise TLS, il doit compilé avec le support TLS. Tous les packages sont disponibles par MariaDB Foundation et MariaDB Corporation.

have_ssl est une variable qui permet de savoir si le serveur prend en charge l'utilisation de TLS :

- Si la valeur est YES, le serveur prend en charge TLS et TLS est activé.
- Si la valeur est DISABLED, le serveur prend en charge TLS, mais TLS n'est pas activé.
- Si la valeur est NO, cela signifie que le serveur n'a pas été compilé avec la prise en charge de TLS, donc TLS ne peut pas être activé.

Qu'est ce que SSL/TLS et comment L’utiliser ?

https://fr.wikipedia.org/wiki/Transport_Layer_Security#:~:text=Le%20protocole%20SSL%20a%20%C3%A9t%C3%A9,Transport%20Layer%20Security%20(TLS)

Le SSL/TLS (Secure Sockets Layer/Transport Layer Security) est un protocole de sécurité utilisé pour établir une communication sécurisée sur Internet. Il permet de sécuriser les données transmises entre un client et un serveur en utilisant un système de cryptage pour protéger les informations sensibles.
Le SSL/TLS fonctionne en établissant une connexion cryptée entre le client et le serveur. Cette connexion cryptée est créée à l'aide d'un certificat SSL/TLS émis par une autorité de certification (CA) de confiance. Ce certificat est une forme d'identification numérique qui permet de vérifier l'identité du serveur et de garantir que la communication est sécurisée.
Le SSL/TLS utilise des algorithmes de chiffrement pour crypter les données transmises entre le client et le serveur. Les algorithmes utilisés dépendent de la version de SSL/TLS et des paramètres de sécurité configurés sur le serveur. Les versions plus récentes de TLS sont plus sécurisées que les versions précédentes, car elles utilisent des algorithmes de chiffrement plus robustes et ont des mécanismes de protection supplémentaires contre les attaques.
Lors d’une connexion sécurisée avec SSL, le serveur demande à ce que le client s'identifie, ainsi le client envoie une copie de son certificat SSL au serveur. Le serveur vérifie s' il peut faire confiance au certificat ssl et si c’est le cas, il envoie un message au client. Le client renvoie une reconnaissance signée par voie numérique pour démarrer une séssion chiffrée ssl.
L'appareil d'un utilisateur visualise la clé publique et l'utilise pour établir des clés de chiffrement sécurisées avec le serveur web. Entre-temps, le serveur web dispose également d'une clé privée qui est gardée secrète ; la clé privée déchiffre les données chiffrées avec la clé publique.

Le système de cryptage du protocole SSL/TLS (Secure Sockets Layer/Transport Layer Security) utilise des algorithmes de chiffrement pour protéger les données transmises entre le client et le serveur. Les algorithmes de chiffrement utilisés peuvent varier en fonction de la version de SSL/TLS et des paramètres de sécurité configurés sur le serveur.
Lorsqu'une connexion SSL/TLS est établie, le client et le serveur négocient un ensemble commun d'algorithmes de chiffrement à utiliser pour protéger la communication. Cette négociation inclut le type d'algorithme de chiffrement symétrique (comme AES ou 3DES) et le type d'algorithme de chiffrement asymétrique (comme RSA ou Diffie-Hellman) qui seront utilisés.
Une fois que les algorithmes de chiffrement sont négociés, les données sont cryptées avant d'être envoyées sur la connexion SSL/TLS. Le chiffrement assure que les données sont illisibles pour toute personne qui intercepte la communication, car elle ne possède pas la clé de déchiffrement.

Normes RFC

https://www.ietf.org/standards/rfcs/

Les RFC (Request for Comments) sont des documents techniques qui décrivent les spécifications, les protocoles, les algorithmes et les bonnes pratiques utilisées dans les domaines des réseaux informatiques et de l'Internet.

En ce qui concerne OpenSSL, les RFC sont importantes car elles définissent les normes de sécurité et les protocoles de communication utilisés dans les applications qui utilisent OpenSSL. Les normes RFC pour OpenSSL incluent :

1. RFC 5280 - Spécifie le format des certificats X.509 utilisés dans OpenSSL pour l'authentification et la sécurité.
2. RFC 5246 - Spécifie le protocole TLS (Transport Layer Security) utilisé dans OpenSSL pour sécuriser les connexions réseau.
3. RFC 5288 - Spécifie les suites de chiffrement utilisées dans TLS, y compris les algorithmes de chiffrement et les modes d'opération.
4. RFC 4492 - Spécifie les algorithmes de chiffrement à courbe elliptique utilisés dans TLS, qui offrent une sécurité plus élevée que les algorithmes de chiffrement classiques.
5. RFC 7515 - Spécifie le format JWS (JSON Web Signature) utilisé dans les applications Web qui utilisent OpenSSL pour la sécurité et l'authentification.

Il est important de suivre les normes RFC lors de la mise en place de la sécurité et de l'authentification dans les applications qui utilisent OpenSSL. Cela garantit une compatibilité avec d'autres applications et une sécurité élevée pour les utilisateurs finaux.

Dans le contexte ici présent nous utilisons la norme RFC 5280 car celle-ci est la plus adaptée au standard X.509 mais aussi car elle compatible avec celui-ci.

Le fichier server-key.pem

Le fichier "server-key.pem" est un fichier de clé privée utilisé par le protocole SSL/TLS pour établir des connexions sécurisées entre un serveur et un client.

La clé privée dans le fichier "server-key.pem" est utilisée pour signer numériquement les données échangées entre le serveur et le client, permettant ainsi de garantir l'authenticité et l'intégrité des données. La clé privée est également utilisée pour décrypter les données chiffrées reçues du client.

Le fichier "server-key.pem" est généralement accompagné d'un fichier de certificat nommé "server-cert.pem". Le certificat contient la clé publique associée à la clé privée dans le fichier "server-key.pem", ainsi que des informations d'identification sur le serveur telles que le nom de domaine et l'organisation émettrice du certificat.

En résumé, le fichier "server-key.pem" est un fichier de clé privée utilisé par le protocole SSL/TLS pour établir des connexions sécurisées entre un serveur et un client, et est généralement accompagné d'un fichier de certificat contenant la clé publique associée et des informations d'identification sur le serveur.

Le fichier "server-key.pem" est utilisé par le serveur pour établir des connexions SSL/TLS sécurisées avec les clients. Voici les étapes générales pour utiliser la clé privée dans le fichier "server-key.pem":

1. *Copiez le fichier "server-key.pem" sur le serveur qui héberge le site web ou l'application qui nécessite une connexion sécurisée.*
2. *Configurez le serveur pour utiliser la clé privée lors de la négociation de la connexion SSL/TLS. Les étapes pour cela dépendent du serveur web utilisé. Par exemple, pour Apache, vous pouvez utiliser la directive "SSLCertificateKeyFile" dans la configuration SSL du serveur pour spécifier le chemin du fichier "server-key.pem".*
3. *Si le fichier "server-key.pem" est accompagné d'un fichier de certificat, configurez également le serveur pour utiliser le fichier de certificat pendant la négociation de la connexion SSL/TLS. Pour Apache, vous pouvez utiliser la directive "SSLCertificateFile" dans la configuration SSL pour spécifier le chemin du fichier de certificat.*
4. *Redémarrez le serveur web pour que les nouvelles configurations soient prises en compte.*

Une fois que la clé privée est configurée et utilisée par le serveur, les clients peuvent établir des connexions sécurisées avec le serveur en utilisant le protocole SSL/TLS. Par exemple, lorsqu'un utilisateur accède à un site web sécurisé, le navigateur du client établit une connexion SSL/TLS avec le serveur en utilisant la clé publique correspondant à la clé privée dans le fichier "server-key.pem". La connexion sécurisée permet de crypter les données échangées entre le client et le serveur, et de garantir l'authenticité et l'intégrité des données.

Vous pouvez générer une nouvelle clé privée dans le format PEM (Privacy Enhanced Mail) utilisé par OpenSSL à l'aide de la commande openssl genpkey. Voici les étapes pour générer une nouvelle clé privée :

1. *Ouvrez une console ou un terminal sur votre système.*
2. *Entrez la commande suivante pour générer une clé privée RSA de 2048 bits dans un fichier nommé "server-key.pem":*
   csharp

openssl genpkey -algorithm RSA -out server-key.pem -aes256

Cette commande utilise l'algorithme RSA avec une longueur de clé de 2048 bits et demande également un mot de passe pour protéger la clé privée. Si vous souhaitez créer une clé privée sans mot de passe, vous pouvez omettre l'option "-aes256".

Si vous souhaitez consulter la clé privée générée, vous pouvez utiliser la commande suivante :
vbnet

openssl pkey -in server-key.pem -text

1. *Cette commande affiche les informations de la clé privée, y compris l'algorithme, la longueur de la clé et le mot de passe (s'il y en a un).*

Une fois que vous avez généré une nouvelle clé privée, vous pouvez la configurer pour être utilisée par le serveur web de votre choix pour établir des connexions sécurisées SSL/TLS.

Configuration des connexions sécurisées :

Vous pouvez configurer les connexions sécurisées en modifiant le fichier de configuration de MariaDB. Vous pouvez ouvrir le fichier en utilisant la commande suivante :

sudo nano /etc/mysql/mariadb.conf.d/50-server.cnf

Ensuite, vous pouvez ajouter les lignes suivantes pour activer SSL :

ssl-ca=/etc/mysql/ssl/ca-cert.pem
ssl-cert=/etc/mysql/ssl/server-cert.pem
ssl-key=/etc/mysql/ssl/server-key.pem
On met ces fichiers dans le .my.cnf

Assurez-vous que les fichiers de certificats et de clés existent dans les emplacements spécifiés. Si vous n'avez pas ces fichiers, vous pouvez les créer en utilisant la commande suivante :

sudo openssl req -newkey rsa:2048 -nodes -keyout /etc/mysql/ssl/server-key.pem -out /etc/mysql/ssl/server-req.pem

sudo openssl x509 -req -in /etc/mysql/ssl/server-req.pem -days 36500 -CA ca-cert.pem -out /etc/mysql/ssl/server-cert.pem

sudo openssl dhparam -out /etc/mysql/ssl/dhparams.pem 2048

sudo cat /etc/mysql/ssl/server-cert.pem /etc/mysql/ssl/server-key.pem > /etc/mysql/ssl/server-ssl.pem

sudo chmod 600 /etc/mysql/ssl/*

Commande pour le certificat auto signé:
openssl req -newkey rsa:2048 -days 365000 -nodes -keyout server-key.pem -out server-req.pem

openssl rsa -in server-key.pem -out server-key.pem

openssl x509 -req -in server-req.pem -days 365000 \
    -CA ca.pem -CAkey ca-key.pem -set_serial 01 \
    -out server-cert.pem

Config client SQL

Il faut activer le paramètre TLS

Du côté de la configuration client il faut avoir un certificat client qu'on obtient avec openSSL

Les commandes utilisé pour l'activation de TLS

[client-mariadb]
...
ssl_cert = /etc/my.cnf.d/certificates/client-cert.pem
ssl_key = /etc/my.cnf.d/certificates/client-key.pem
ssl_ca = /etc/my.cnf.d/certificates/ca.pem
Les commandes utilisé pour l'activation de TLS en mode bidirectionnel

TLS bidirectionnel signifie que le client et le serveur fournissent tous deux une clé privée et un certificat X509

[client-mariadb]
...
ssl_cert = /etc/my.cnf.d/certificates/client-cert.pem
ssl_key = /etc/my.cnf.d/certificates/client-key.pem
ssl_ca = /etc/my.cnf.d/certificates/ca.pem
ssl-verify-server-cert
Les commandes utilisé pour l'activation de TLS en mode unidirectionnel

TLS unidirectionnel signifie que seul le serveur fournit une clé privée et un certificat X509.

[client-mariadb]
...
ssl_ca = /etc/my.cnf.d/certificates/ca.pem
ssl-verify-server-cert

Pour configurer le client pour qu'il utilise TLS
Il faut éditer dans le répertoir du client son fichier my.cnf qui se situe dans /etc/mysql/my.cnf

https://mariadb.com/kb/en/configurin

Location	Scope
`/etc/my.cnf`	Global
`/etc/mysql/my.cnf`	Global
`$MARIADB_HOME/my.cnf`	Server
`$MYSQL_HOME/my.cnf`	Server
defaults-extra-file	File specified with `--defaults-extra-file`, if any
`~/.my.cnf`	User

Comment sécuriser une connexion personnalisé

on cherche à obtenir : Le client et le serveur s’authentifie mutuellement via un certificat d’autorité auto-signé

client
serveur

1-requête à une ressource protégée
2-transmet le certificat serveur
3- transmet le certificat client
4-vérifie le certificat serveur
5-vérifie le certificat client
6- accès à la ressource protégée

g-mariadb-with-option-files/

nom du adduser : serv
mdp : azerty

Certificat d'autorité :