Base de données

Cours 13 - Sécurité, sauvegarde et restauration

Table des matières

Qu'est-ce que le chiffrement?

Le chiffrement est le processus de conversion de données ou d'informations en un code secret pour empêcher les accès non autorisés.
Il utilise des algorithmes mathématiques pour transformer les données en une forme illisible, appelée texte chiffré, qui ne peut être déchiffrée qu'avec une clé spécifique.

Pourquoi le chiffrement est-il important?

- Protection des données sensibles
- Conformité aux réglementations (RGPD, HIPAA, etc.)
- Prévention des accès non autorisés
- Maintien de la confidentialité et de l'intégrité des données
- Protection contre les fuites de données
- Sécurisation des communications réseau

Type de chiffrement

- Chiffrement des mots de passe (Password Encryption)
- Chiffrement pour des colonnes spécifiques (Encryption For Specific Columns)
- Chiffrement des partitions de données (Data Partition Encryption)
- Chiffrement des données sur le réseau (Encrypting Data Across A Network)
- Authentification SSL des hôtes (SSL Host Authentication)
- Chiffrement côté client (Client-Side Encryption)

Chiffrement des mots de passe

Le chiffrement des mots de passe est une mesure de sécurité essentielle pour protéger les informations d'identification des utilisateurs d'une base de données.

Les mots de passe des utilisateurs de la base de données sont stockés sous forme de hachages déterminés par le paramètre `password_encryption`, de sorte que l'administrateur ne puisse pas déterminer le mot de passe réel attribué à l'utilisateur.

Chiffrement des mots de passe

PostgreSQL prend en charge plusieurs méthodes de chiffrement des mots de passe, notamment:

- **MD5**: Méthode de hachage courante, mais moins sécurisée que SCRAM-SHA-256 (déprécié).
- **SCRAM-SHA-256**: Méthode plus sécurisée recommandée pour le chiffrement des mots de passe.

Par défaut, PostgreSQL utilise SCRAM-SHA-256 pour le chiffrement des mots de passe.

Chiffrement des mots de passe

```sql
-- When you provide a password directly in the CREATE ROLE statement,
-- PSQL will hash it using the method configured by password_encryption
CREATE USER temp_user WITH PASSWORD 'temp_password123';

-- To create a user with a SCRAM-SHA-256 hashed password,
--  you need to provide the SCRAM-SHA-256 hash directly.
CREATE USER user_hashed WITH
    LOGIN
    PASSWORD 'SCRAM-SHA-256<your_hash_here>';
```

Listing: https://www.postgresql.org/docs/current/sql-createrole.html

	usename name	passwd text
1	temp_user	SCRAM-SHA-256$4096:TDox892eRaq+npwykINJQA==$5317sMd/1Lr0l9M7QNJHYQQqO8YcwevdmjC2aLa1/PI=:FRkcKuPnKzRn6nneBNAY4PPraXcX0OPHXtXN...
2	user_hashed	SCRAM-SHA-256$4096:G7ciDbMUZAhFIVjoli2R0Q==$RARFVMe1miB9v014RmNyrVkfjvsksZ1MiZagQiB+ivs=:nANkINTRVZn4iGdmFhrUu4p6XqvJ/xqkvgGBKu4k...

Figure: https://www.postgresql.org/docs/current/sql-createrole.html

Chiffrement pour des colonnes spécifiques

Le chiffrement au niveau des colonnes permet de protéger des données sensibles dans des colonnes spécifiques d'une table.

Le client fournit la clé de déchiffrement, et les données sont déchiffrées sur le serveur avant d'être envoyées au client. Les données et la clé de déchiffrement sont temporairement présentes sur le serveur lors du déchiffrement, ce qui crée un court risque d'interception par une personne ayant un accès complet au serveur, comme un administrateur système.

Le module `pgcrypto` permet de stocker certains champs de manière chiffrée. Il offre des fonctions pour chiffrer et déchiffrer les données, ainsi que pour générer des hachages cryptographiques.

Chiffrement pour des colonnes spécifiques

```
1  -- Install pgcrypto extension for encryption functions
2  CREATE EXTENSION IF NOT EXISTS pgcrypto;
3
4  CREATE TABLE sensitive_data (
5  id serial primary key,
6  encrypted_field bytea not null
7  );
8
9  -- PGP_SYM_ENCRYPT() for symmetric-key encryption
10 -- the same key is used for both encryption and decryption
11 INSERT INTO sensitive_data (id, encrypted_field)
12 VALUES (1, PGP_SYM_ENCRYPT('secret_value', 'your_encryption_key'));
```

Listing: https://www.postgresql.org/docs/current/sql-createrole.html

Chiffrement pour des colonnes spécifiques

Si un utilisateur tente d'accéder aux données chiffrées sans la clé appropriée, il ne pourra pas les lire.

Query | Query History

```
1  select id, encrypted_field from sensitive_data;
```

Data Output | Messages | Notifications

	id [PK] integer	encrypted_field bytea
1	1	[binary data]

Figure: `https://www.postgresql.org/docs/current/pgcrypto.html`

Chiffrement pour des colonnes spécifiques

En déchiffrant les données avec `PGP_SYM_DECRYPT()` et la bonne clé:

Query | Query History

```
1 SELECT id, PGP_SYM_DECRYPT(encrypted_field, 'your_encryption_key')
2 AS decrypted_value
3 FROM sensitive_data;
```

Data Output | Messages | Notifications

	id [PK] integer	decrypted_value text
1	1	secret_value

Figure: `https://www.postgresql.org/docs/current/pgcrypto.html`

Chiffrement pour des colonnes spécifiques

Il est également possible de chiffrer des données dans une table en utilisant `crypt()`:

```
1  CREATE EXTENSION IF NOT EXISTS pgcrypto;
2
3  -- Update existing passwords to be hashed using crypt function
4  UPDATE user
5  SET password = crypt(password, gen_salt('sha256crypt'));
6
7  -- To verify a password later, you can use:
8  SELECT * FROM user
9  WHERE username = 'john'
10 AND password = crypt('entered_password', password);
```

Listing: `https://www.postgresql.org/docs/current/sql-createrole.html`

Cependant, les données chiffrées avec `crypt()` ne peuvent pas être déchiffrées pour retrouver la valeur originale.

Chiffrement des partitions de données

Le chiffrement des partitions de données consiste à chiffrer l'ensemble des fichiers de la base de données au niveau du système de fichiers.

Le chiffrement se fait au niveau du système d'exploitation (ex: BitLocker sur Windows, FileVault sur Mac, LUKS sur Linux).

C'est une protection contre le vol physique. Cependant, une fois que l'ordinateur est allumé et que le disque est "déverrouillé" (monté), le système d'exploitation voit toutes les données en clair. Si un pirate a accès au serveur en marche, ce chiffrement ne l'arrête pas.

Chiffrement des données sur le réseau

Le chiffrement de données sur le réseau crée un "tunnel" sécurisé entre le client et le serveur pour que personne ne puisse intercepter les données en transit.

En utilisant des protocoles comme SSL/TLS ou GSSAPI, toutes les données échangées sur le réseau sont chiffrées, assurant la confidentialité et l'intégrité des informations.

C'est une protection essentielle pour toute communication sur internet.

Authentification SSL des hôtes

L'authentification SSL des hôtes permet de vérifier l'identité du serveur PostgreSQL auquel le client se connecte.

Le client et le serveur s'échangent et vérifient mutuellement leurs certificats SSL, ainsi le client peut s'assurer qu'il communique avec le serveur légitime et non avec un imposteur.

Cela protège contre les attaques de type "Man in the Middle" (homme du millieu) où un attaquant pourrait se faire passer pour le serveur et voler les données.

Chiffrement côté client

Le chiffrement côté client signifie que les données sont chiffrées avant d'être envoyées au serveur de base de données.

Les données sont chiffrées sur le client avant d'être envoyées au serveur. Le serveur ne voit et ne stocke que des données chiffrées. Pour les relire, le client doit les retélécharger et les déchiffrer sur sa machine avec sa clé.

Ainsi, même si un attaquant accède à la base de données, il ne pourra pas lire les données sans la clé de déchiffrement détenue par le client.

Sauvegarde

La sauvegarde des données est essentielle pour protéger contre la perte de données due à des pannes matérielles, des erreurs humaines, ou des catastrophes naturelles

SQL DUMP

La commande `pg_dump` est un utilitaire de sauvegarde pour PostgreSQL qui permet de créer une sauvegarde logique d'une base de données.

Elle génère un fichier texte contenant des commandes SQL nécessaires pour recréer la base de données, y compris les schémas, les tables, les données, et les objets associés.

Cette méthode est flexible et permet de restaurer la base de données sur n'importe quel serveur PostgreSQL compatible.

SQL DUMP - Exemple

Voici la syntaxe de base de la commande pg_dump:

```
pg_dump -h <host> -U <user> -d <db_name> [-t <table_name>]
        -f <output_file>.sql

-- or simply
pg_dump db_name > output_file.sql
```

Listing: https://www.postgresql.org/docs/current/app-pgdump.html

File System Level Backup

La sauvegarde au niveau du système de fichiers consiste à copier directement les fichiers de la base de données au niveau du système de fichiers.

Cette méthode est rapide et efficace, surtout pour les grandes bases de données, car elle évite la surcharge de l'exportation des données en SQL.

Cependant, elle nécessite que la base de données soit arrêtée ou que des mécanismes de journalisation soient en place pour garantir la cohérence des données.

File System Level Backup - Exemple

Voici un exemple de commande pour effectuer une sauvegarde au niveau du système de fichiers:

```
1  -- Stop PostgreSQL service
2  sudo systemctl stop postgresql
3
4  -- Simple copy of a database to a file
5  sudo cp -rp /var/lib/postgresql/data /backup/postgres_backup
6
7  -- or with rsync
8  sudo rsync -av /var/lib/postgresql/data/ /backup/postgres_backup/
```

Listing: `https://www.postgresql.org/docs/current/backup-file.html`

Restauration

La restauration permet de récupérer les données après une perte, en utilisant les sauvegardes créées précédemment

SQL DUMP

La restauration d'une base de données à partir d'un fichier SQL généré par `pg_dump` se fait en utilisant la commande `psql`.

Cette commande lit le fichier SQL et exécute les commandes qu'il contient pour recréer la base de données, y compris les schémas, les tables, et les données.

Il est important de s'assurer que la base de données cible existe avant de lancer la restauration.

SQL DUMP - Exemple

Voici la syntaxe de base de la commande `psql`:

```
1 psql -h <host> -U <user> -d <db_name> -f <input_file>.sql
2
3 -- or simply
4 psql db_name < input_file.sql
```

Listing: `https://www.postgresql.org/docs/current/app-pgdump.html`

File System Level Backup

La restauration au niveau du système de fichiers consiste à copier les fichiers de sauvegarde directement dans le répertoire de données de PostgreSQL.

Avant de restaurer, il est crucial d'arrêter le serveur PostgreSQL pour éviter toute corruption des données.

Après avoir copié les fichiers, le serveur peut être redémarré, et la base de données sera restaurée à l'état qu'elle avait au moment de la sauvegarde.

File System Level Backup - Exemple

Voici un exemple de commande pour effectuer une restauration au niveau du système de fichiers:

```
1  -- Stop the PostgreSQL server before restoring
2  sudo systemctl stop postgresql
3
4  -- Restore the database files from the backup
5  sudo cp -rp /backup/postgres_backup /var/lib/postgresql/data
6
7  -- Then, ensure the correct ownership and permissions
8  sudo chown -R postgres:postgres /var/lib/postgresql/data
9  sudo chmod -R 700/var/lib/postgresql/data
10
11 --- Start the PostgreSQL server after restoration
12 sudo systemctl start postgresql
```

Listing: https://www.postgresql.org/docs/current/backup-file.html

Bibliographie

- https://www.postgresql.org/docs/current/encryption-options.html
- https://www.postgresql.org/docs/current/pgcrypto.html
- https://www.postgresql.org/docs/current/ssl-tcp.html
- https://www.postgresql.org/docs/current/backup.html
- https://www.postgresql.org/docs/current/backup-dump.html
- https://www.postgresql.org/docs/current/app-pgdump.html
- https://www.postgresql.org/docs/current/backup-file.html