L'injection SQL

S’applique à :

- SQL Server (toutes les versions prises en charge)

- Azure SQL Database

- Azure SQL Managed Instance

- Azure Synapse Analytics Analytics

- Platform System (PDW)

# Le fonctionnement

Les injections SQL => insertions directes de code dans les entrées utilisateur concaténées avec des commandes SQL et exécutées.

Des attaques par injection moins directes insèrent un code malveillant dans les chaînes destinées à être stockées dans une table ou en tant que métadonnées.

Lorsque les chaînes stockées sont ensuite concaténées dans une commande SQL dynamique, le code nuisible est exécuté.

Injection = terminer prématurément une chaîne de texte et ajouter une nouvelle commande.

Terminer la chaîne injectée par une marque de commentaire « -- ». Le texte qui vient à la suite est ignoré au moment de l'exécution.

* « ; » => fin de requête
* « -- » commentaire
* Si le code modifié est syntaxiquement correct, il sera exécuté par le serveur.
* Exécution => Sélection pour OrdersTable où ShipCity est 'Redmond' puis suppression de OrdersTable.

Code syntaxiquement correct ? Indétectable.

/!\Vérification des entrées utilisateur /!\

## Bonnes pratiques

* /!\Données utilisateurs reçues.
* Prévoir
  + Insertion fichier JPEG au lieu d'un texte
  + Insertion d'une instruction `DROP TABLE` dans un champ texte
  + ...
* Tester taille et type des données reçues
* Tester le contenu des variables pour n'accepter que les valeurs attendues.
* Rejeter les entrées contenant des données binaires, des séquences de caractères d'échappement et des caractères de commentaire

Peut empêcher l'injection de scripts et protéger contre l'utilisation de dépassements de mémoire tampon.

* Si docs XML, validation les données basées sur les schémas.
* /!\`Transact-SQL` dans un Input
* Utilisation de procédures stockée
* Validation des données avant d'entrer dans les zones de confiance
* Plusieurs niveaux de validation :

Utilisateurs malintentionnés ≠ pirates déterminés.

Validation dans les Input et à toutes les étapes du programme

La validation des données dans une application côté client peut empêcher l'injection de scripts simples.

Si le niveau suivant considère que son entrée a déjà été validée, un utilisateur malveillant capable de contourner un client peut disposer d'un accès complet à un système.

Pas de concaténation d'une entrée utilisateur => point d'entrée principal

# Utilisation de paramètres SQL de type sécurisé

SQL Server possède une collection `Parameters` qui fournit le contrôle le type et valide la longueur.

=> Les entrées traitées comme des string pas comme du codeéxecutable

=> Permet des contrôles de type et de longueur.

=> Déclenche une exception si la valeur n'est pas dans les limites.

# Utilisation d'entrées paramétrables avec des procédures stockées

Les injections SQL peuvent cibler les procédures stockées notamment si elles utilisent des entrées non filtrées.

Lors de l'utilisation de pocédures stockées, il est impératif d'utiliser des paramètres en entrée.

# Utilisation de la collection Parameters avec des instructions SQL dynamiques

Si l'utilisation de procédure stockée est impossible, il est quand même possible d'utiliser des paramètres.

## Filtrage des entrées

Le filtrage des entrées peut être utile pour éviter des injections SQL par la suppression des caractères d'échappement.

Peu fiable car beaucoup de caractères peuvent poser des problèmes.

# Clauses LIKE

Même avec `LIKE`, les caractères génériques devront quand même être séparés par des caractères d’échappement

# Examen du code à la recherche d'injection SQL

Examination du code qui appelle `EXECUTE`, `EXEC` ou `sp\_executesql`.

Utilsation de reqêtes pour identifier des procédures quicontiennent ces instructions.

Recherche de 1, 2, 3 ou 4 espaces après les mots `EXECUTE` ou `EXEC`.

# Enveloppement de paramètres avec `QUOTENAME()` et `REPLACE()`

Dans chaque procédure stockée vérification que les variables utilisées dans du Transact-SQL dynamique sont traitées correctement.

Les données issues des paramètres d'entrée de la procédure stockée ou lues à partir d'une table doivent être enveloppées dans QUOTENAME() ou REPLACE().

La valeur de @variable transmise à QUOTENAME() est de type sysname et sa longueur maximale de 128 caractères.

# Injection activée par la troncature des données

Une variable Transact-SQL dynamique affectée à une variable est tronquée si elle est supérieure à la mémoire tampon allouée à cette variable.

Un attaquant capable de forcer la troncature en transmettant une chaine plus longue peut manipuler le résultat.

La procédure stockée créée par le script suivant court un risque d'injection permis par la troncature.

En transmettant 154 caractères dans une mémoire tampon de 128 caractères, un attaquant peut définir un nouveau mot de passe sans connaître l’ancien.

C'est pourquoi il faut utiliser une mémoire tampon volumineuse pour une variable de commande ou pour exécuter le Transct-SQL dans l'instruction EXECUTE.

Connexion over SSL

SSL => Secure Sockets Layer

TLS => Transport Layer Security (Successeur)

Protocoles de sécurisation des écahnges par réseau informatiques (internet)

Fonctionnement client-serveur, sécurisation par :

* Autentification du serveur
* Confidentialité des Données échangées (session chiffrée)
* Intégrité des données échangées
* Authentification du client souvent assurée par la couche applicative (Optionnel)

Mise en oeuvre falité car juste à "ajouter" le SSL/TLS pour passer de HTTP à HTTPS

mTLS => mutual TLS

TLS + Autentification obligatoire du client

Type de connexion sécurisée :

* Utilisateur + mot de passe hashé
* unix\_socket (plugin) => connexion basée sur les droits de la session (linux)

tcp + transport

trames

login via session user unix\_socket plugin

Sources

* Injections SQL : [Microsoft](https://learn.microsoft.com/fr-fr/sql/relational-databases/security/sql-injection?view=sql-server-ver16)
* Plugins : [MySQL](https://dev.mysql.com/doc/refman/8.0/en/authentication-plugins.html)
* SSL / TLS / mTLS : [medium](https://medium.com/double-pointer/ssl-vs-tls-vs-mtls-f5e836fe6b6d)