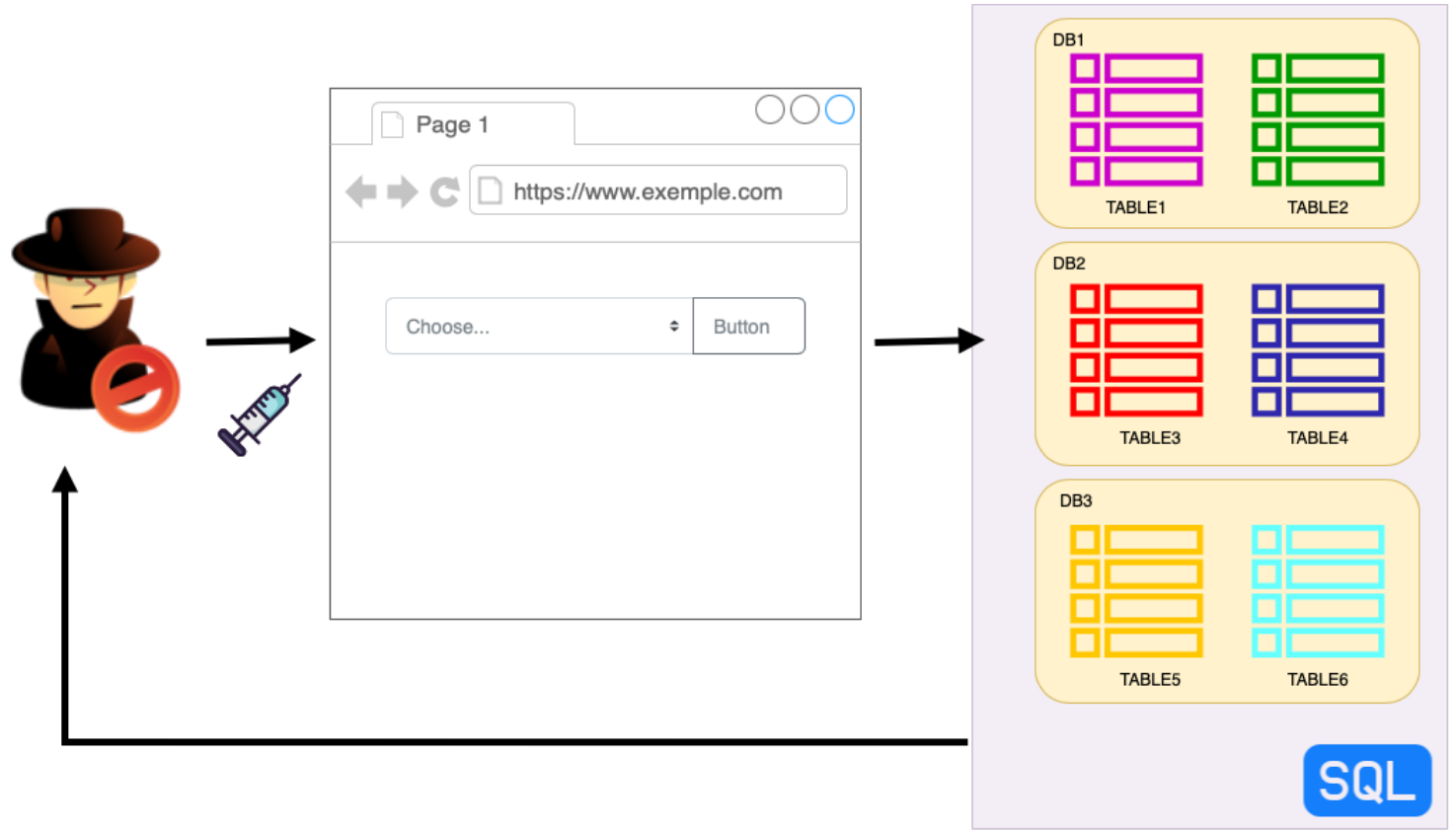
SQL Injection

INJECTIONS SQL

L'injection SQL est une technique d'attaque qui exploite les vulnérabilités dans une application web en manipulant les requêtes SQL. Les attaquants peuvent utiliser l'injection SQL pour interroger, modifier ou même supprimer les données d'une base de données. Ils peuvent également modifier le file system et exécuter des commandes sur l'OS.



Comment fonctionne l'injection SQL
L'injection SQL se produit lorsque les entrées de l'utilisateur sont insérées directement dans une requête SQL sans être correctement nettoyées ou échappées. Par exemple, considérez le code suivant :

```
let username = req.body.username;
let password = req.body.password;
let query = `SELECT * FROM users WHERE username = '${username}' AND
password = '${password}'`;
```

Un attaquant pourrait fournir une valeur pour username comme admin' --, ce qui transformerait la requête en : **`SELECT * FROM users WHERE username = 'admin' --' AND password = "`**

Le -- est un commentaire en SQL, donc tout ce qui suit est ignoré. Cela permet à l'attaquant de se connecter en tant qu'administrateur sans connaître le mot de passe.

Il existe plusieurs types d'injections SQL, chacun ayant ses propres caractéristiques et méthodes d'exploitation. Voici quelques-uns des types les plus courants :

On peut les diviser en 3 principales catégories :

- **In-band injection** :

C’est l’injection classique. L’attaquant lance une attaque et obtient un résultat à travers le même canal de communication.

- **Inferential ou blind injection** :

C’est une technique d'injection SQL où les données de la base de données ne sont pas directement transmises à l'attaquant, et l'attaquant n'est pas capable de voir le résultat de l'attaque directement dans la sortie de sa requête. Au lieu de cela, l'attaquant est capable d'obtenir les informations nécessaires en envoyant des requêtes qui provoquent un changement de comportement de la base de données ou de l'application. En observant ces changements de comportement, l'attaquant peut en déduire certaines informations - d'où le nom "inferential" ou "blind".

- **Out-of-band injection** :

C’est une technique d'injection SQL qui permet à un attaquant d'exfiltrer des données en utilisant une communication différente de celle utilisée pour l'injection elle-même, par exemple en utilisant le DNS ou HTTP(S) comme canaux de transmission des données. Par exemple, l'attaquant peut provoquer l'envoi d'une requête DNS vers un serveur DNS qu'il contrôle, avec les données exfiltrées incorporées dans le nom de domaine demandé. Lorsque le serveur DNS reçoit la requête, il peut enregistrer les données exfiltrées.

1.1 Injection SQL basée sur l'erreur (Error-Based SQLi) :
Cette technique consiste à provoquer une erreur dans une requête SQL afin de récupérer des informations utiles à partir du message d'erreur renvoyé par la base de données.

Prenons un exemple simple. Supposons que vous ayez une application web qui utilise la requête SQL suivante pour afficher les détails d'un utilisateur :
`SELECT * FROM users WHERE id = [ID]`

Ici, [ID] est remplacé par l'ID de l'utilisateur que l'application récupère à partir de l'URL (par exemple, www.example.com/user?id=1).
Un attaquant pourrait manipuler l'URL pour injecter une autre requête SQL qui provoque une erreur.
Par exemple, l'attaquant pourrait changer l'URL en www.example.com/user?id=1'.
Cela modifierait la requête SQL en : `SELECT * FROM users WHERE id = 1'`
La requête est maintenant mal formée car il y a une apostrophe supplémentaire à la fin, ce qui provoque une erreur SQL. Si l'application renvoie le message d'erreur SQL complet à l'utilisateur, l'attaquant pourrait obtenir des informations sur la structure de la base de données.

Pour prévenir ce type d'attaque, il est recommandé de ne jamais renvoyer de messages d'erreur SQL détaillés à l'utilisateur.

1.2 Injection SQL basée sur l'union (Union-Based SQLi) : Cette technique utilise l'opérateur UNION SQL pour combiner la réponse de la requête originale avec les résultats d'autres requêtes injectées. Cela permet à l'attaquant de récupérer des informations supplémentaires à partir de la base de données.

Voici un exemple simple. Supposons que vous ayez une application web qui utilise la requête SQL suivante pour afficher les détails d'un utilisateur : **`SELECT * FROM users WHERE id = [ID]`**

Ici, [ID] est remplacé par l'ID de l'utilisateur que l'application récupère à partir de l'URL (par exemple, www.example.com/user?id=1).
Un attaquant pourrait manipuler l'URL pour injecter une autre requête SQL. Par exemple, l'attaquant pourrait changer l'URL en www.example.com/user?id=1 UNION SELECT * FROM admin.
Cela modifierait la requête SQL en :
`SELECT * FROM users WHERE id = 1 UNION SELECT * FROM admin`

Cette requête renverrait d'abord les détails de l'utilisateur avec l'ID 1, puis elle renverrait tous les détails de la table admin. Si l'application affiche les résultats de cette requête sur une page web, l'attaquant pourrait voir les détails de tous les administrateurs.
Il est important de noter que cette attaque ne fonctionnerait que si les deux requêtes SELECT renvoient le même nombre de colonnes. Sinon, la base de données renverrait une erreur.

Pour prévenir ce type d'attaque, il est recommandé d'utiliser des requêtes paramétrées ou préparées, qui garantissent que les entrées de l'utilisateur sont toujours traitées comme des données littérales, et non comme une partie de la requête SQL.

Inferential ou blind injection

L'injection SQL aveugle, ou Blind SQLi, est une sous-catégorie d'attaques d'injection SQL. Dans une injection SQL classique, l'attaquant peut voir le résultat de leur injection directement dans la réponse de l'application. Par exemple, si l'attaquant injecte ' OR '1'='1, ils peuvent voir toutes les données de la table si l'application affiche les résultats de la requête SQL.

Cependant, dans une injection SQL aveugle, l'application ne renvoie pas les résultats de la requête SQL dans sa réponse. Cela peut être dû à diverses raisons, par exemple, l'application ne renvoie que des réponses booléennes (comme "Connexion réussie" ou "Échec de la connexion"), ou l'application ne renvoie pas de réponse du tout.

Dans ce cas, l'attaquant doit utiliser d'autres techniques pour extraire des informations. Voici deux techniques courantes :

1. **Injection SQL aveugle basée sur le contenu (Content-Based Blind SQLi).**
2. **Injection SQL aveugle basée sur le temps (Time-Based Blind SQLi).**

Ces techniques peuvent être très lentes, car l'attaquant doit injecter une requête pour chaque bit d'information qu'ils veulent extraire. Cependant, elles peuvent être très puissantes si l'application est vulnérable à l'injection SQL et ne renvoie pas les résultats de la requête SQL dans sa réponse.

Voici des exemples pour chaque type d'injection SQL aveugle :

1. **Injection SQL aveugle basée sur le contenu (Content-Based Blind SQLi)** :

Supposons que vous ayez une application web qui permet aux utilisateurs de se connecter avec un nom d'utilisateur et un mot de passe. Lorsqu'un utilisateur tente de se connecter, l'application exécute la requête SQL suivante :

```
SELECT * FROM users WHERE username = '[username]' AND password = '[password]'
```

où [username] et [password] sont remplacés par le nom d'utilisateur et le mot de passe fournis par l'utilisateur. Si la requête renvoie au moins une ligne, l'application renvoie "Connexion réussie". Sinon, elle renvoie "Échec de la connexion".

Un attaquant pourrait injecter la requête suivante comme nom d'utilisateur :

```
admin' AND (SELECT password FROM users WHERE username = 'admin') LIKE 'a%
```

Si l'application renvoie "Connexion réussie", l'attaquant peut en déduire que le mot de passe de l'administrateur commence par 'a'. L'attaquant peut répéter ce processus pour chaque lettre du mot de passe.

2.Injection SQL aveugle basée sur le temps (Time-Based Blind SQLi) :

Supposons que vous ayez la même application web que dans l'exemple précédent. Un attaquant pourrait injecter la requête suivante comme nom d'utilisateur :

```
admin' AND IF((SELECT password FROM users WHERE username = 'admin') LIKE 'a%', SLEEP(10), 'false')
```

Si l'application met plus de temps que d'habitude pour répondre, l'attaquant peut en déduire que le mot de passe de l'administrateur commence par 'a'. L'attaquant peut répéter ce processus pour chaque lettre du mot de passe.

Ces exemples sont simplifiés pour illustrer les concepts. Dans une situation réelle, un attaquant utiliserait probablement des outils automatisés pour effectuer ces attaques, car elles peuvent nécessiter des milliers de requêtes pour extraire des informations significatives.

5. Injection SQL basée sur l'OUTFILE (Out-of-band SQLi) : Cette technique est utilisée lorsque l'attaquant est capable d'exploiter une fonctionnalité de la base de données qui lui permet d'enregistrer les résultats d'une requête dans un fichier, et qu'il a la possibilité de récupérer ce fichier.

L'application peut ne pas contenir les résultats de la requête, soit parce que l'application ne les affiche pas, soit parce que la requête injectée renvoie trop de données pour être affichées dans la réponse.
Dans ces cas, l'attaquant peut utiliser l'injection SQL basée sur l'OUTFILE pour enregistrer les résultats de la requête dans un fichier sur le système de fichiers du serveur, puis récupérer ce fichier en utilisant une autre vulnérabilité, comme une vulnérabilité de traversée de répertoires.
Voici un exemple d'injection SQL basée sur l'OUTFILE :

```
'; SELECT * INTO OUTFILE '/var/www/html/dump.txt' FROM users; --
```

Dans cet exemple, l'attaquant injecte une requête SQL qui enregistre toutes les données de la table users dans un fichier nommé dump.txt dans le répertoire /var/www/html/. Si le serveur est mal configuré et permet à la base de données d'écrire dans ce répertoire, et si l'attaquant peut accéder à ce fichier via le web (par exemple, en visitant http://example.com/dump.txt), alors l'attaquant peut récupérer toutes les données de la table users.

Prévention des injections SQL

Il existe plusieurs techniques pour prévenir les injections SQL :

1. **Utiliser des requêtes paramétrées ou préparées** : Ces requêtes garantissent que les entrées de l'utilisateur sont toujours traitées comme des données littérales, et non comme une partie de la requête SQL.

2. **Valider et échapper les entrées de l'utilisateur** : Toutes les entrées de l'utilisateur doivent être validées et échappées avant d'être utilisées dans une requête SQL.

3. **Utiliser un ORM (Object-Relational Mapping)** : Les ORM peuvent aider à prévenir les injections SQL en encapsulant les requêtes SQL brutes.

4. **Utiliser des privilèges de base de données limités** : L'application doit se connecter à la base de données avec le minimum de privilèges nécessaire.

5. **Utiliser un pare-feu d'application web (WAF)** : Un WAF peut aider à bloquer les attaques d'injection SQL en identifiant et en bloquant les requêtes malveillantes.