Injections SQL

Détails d’une faille dangeureuse

Sommaire

- Présentation
- Méthodes d’exploitation
- Méthodes de défense

Présentation

- SQL : Structured Query Language
- Métalangage de manipulation de données par requêtes
- Nombreux langages dessus : MySQL, MariaDB, SQLite,...

```
MariaDB [test]> select * from mytable;
+----+--------+
| id | name   |
+----+--------+
|  1 | Alexis |
+----+--------+
1 row in set (0.00 sec)
```

Présentation

- Attaque par injection : détourner une requête en corrompant les données envoyées
- Différents types : Injection SQL, NoSQL, XPath, LDAP, commande,... : dépend du langage de requête

```
$query = "SELECT * FROM mytable WHERE user = '" . $_POST["user"] . "' AND password = '" . $_POST["password"] ."';" ;

avec $_POST["user"] = "' OR 1=1; --"

devient

$query = "SELECT * FROM mytable WHERE user='' OR 1=1; --"; (le reste est en commentaire)
```

Méthodes d’exploitation

- Différentes méthodes :
  - Selon but recherché :
    - Authentification corrompue
    - Récupération de données
    - Corruption de la base de données
  - Selon données récupérées :
    - Visibilité des données récupérées
    - Injections en aveugle
    - Time-based injections
    - Error-based injections
  - Selon protections en place :
    - Contournement de filtre
    - Troncation

  Et bien d’autres critères encore...

Méthodes d’exploitation

- Repérer un vecteur d’injection :
  - Tenter de réaliser une erreur de syntaxe : ajout de ', ", ;, #,...
  - Tenter de découvrir le langage utilisé (variantes des commentaires # ou --,...)
  - Comprendre la formulation de la requête (combien de champs sélectionnés, conditions,...)
  - Trouver les tables INFORMATION_SCHEMA (toutes les infos de la BDD)
- Décider du type d’attaque :
  - Avec ou sans récupération de données
  - Quantité d’informations récupérables : en clair, en aveugle, en time-based

Méthodes d’exploitation

- Corruption d’authentification :
  - Se connecter en bypassant l’authentification par mot de passe
  - Conditions mises à vrai peu importe le mot de passe
  - Utile pour se connecter au compte admin

```
$query = "SELECT * FROM mytable WHERE user = '" . $_POST["user"] . "' AND password = '" . $_POST["password"] ."';" ;

avec $_POST["user"] = "' OR 1=1; --"

devient

$query = "SELECT * FROM mytable WHERE user='' OR 1=1; --"; (le reste est en commentaire)
```

Méthodes d’exploitation

- UNION-based :
  - Extraire des données dans un champ non protégé
  - Principe :

SELECT XXX,XXX FROM XXX WHERE cond='' AND 1=0

UNION SELECT YYY,YYY FROM YYY;--

  - Utiliser la requête pour chercher d’autres données
  - Trouver le nombre de champs sélectionnés, le nom des tables et des champs

Méthodes d’exploitation

- Blind injection :
  - Seule information renvoyée : true or false
  - Deviner petit à petit les informations
  - Tests binaires : =, !=, <, >, IN, BETWEEN,...
  - Ex:

    SELECT * FROM users WHERE name=’’ OR substring(name,1,1)=’a’ ;--

  - Nécessite un script d’automatisation d’envoi
  - PAS DU TOUT DISCRET !

Méthodes d’exploitation

- Time-based :
  - Blind injection sans renvoi d’information
  - Utiliser le temps de réponse come information
  - Commande SLEEP(n) fait une pause de n secondes
  - Exemple :

SELECT * FROM users WHERE name=’’ OR substring(name,1,1)=’a’ OR SLEEP(1)

  - Commande SLEEP activée si et seulement si name n’est pas vide ET sa première lettre n’est pas ‘a’.
  - Moyens plus rapides, cf internet

Méthodes de protection

- Filtres:
  - Échapper les caractères dangereux : ',",#,;,...
  - En PHP : htmlentities($string) remplace caractères par code HTML

- Requêtes préparées :
  - Deux étapes : forger la requête puis l'envoyer
  - But : variables ajoutées après analyse syntaxique de la requête
  - En PHP : prepare("SELECT ? FROM table where id=?", $var1,$var2)

Conclusion

- Faille DANGEREUSE
- BDD = des informations ULTRA sensibles
- Beaucoup de possibilités
- Parfois beaucoup d’ingéniosité
- Des moyens SIMPLES de protection

Conclusion

- Beaucoup de vecteurs d’attaque
-