SQLインジェクション

SQLインジェクション

不正なSQL文を生成し、実行させる攻撃。情報の漏洩、内容改ざん、認証回避、DBサーバのファイル読み出し・書き込み、不正なプログラムの実行などの影響がある。

ブラインドSQLインジェクション　・・・　ユーザやテーブル名などDBMS (DataBase Manegement System)に関する情報を盗み出すSQLインジェクションのこと。

対策　・・・　静的**プレースホルダ**を利用する

SQLインジェクションによる漏えい例

1. エラーメッセージによる情報漏えい？

   ～?author='+ and **extractivalue**(0,select+**concat('$',id,':',pwd)**+from+users+limit+0,1))+**%23**

   ↓ extractivalue()・・・無効な構文としてエラーを表示させるために使用している。

   concat()・・・複数の文字列を連結する。この場合は**$id:pwd**を生成する。

   ～?author='' and extractivalue(0,select concat('$',id,':',pwd) from users limit 0,1)) #

   idとpwdを先頭の１件のみ文字列($id:pwd)にして表示する。

2. union selectを用いたもの

   union　・・・　**複数のselect文**を統合して表示する仕組み

```
～?author='+ union+select+id,pwd,name,addr,null,null+from+users –+       //入力する文字列、--はコメント化
      ↓
～?author='' union select id,pwd,name,addr,null,null from users
```

   null・・・プログラムで出力されるフィールド数を合わせるために使用している。

認証回避

ログイン認証などの回避として、次のような入力が使用される。

<例>ユーザ名がわかっている場合

ユーザ名・・・入力

パスワード・・・　' or 'a'='a　　または　' or 1 = 1

<例>ユーザ名を「tanaka」、パスワードを「**'or 'a'='a**」と入力した場合

```
$sql = "select * from users where id = '". $_POST['namae']  . "'  and  pwd = '" . $_POST['pwd'] ."'";        //元のプログラム
      ↓
$sql = "select * from users where id = 'tanaka' and pwd ='' or 'a' = 'a'";
```

改ざん

updateコマンドを利用してデータの改ざんを行う。?チェック

<例>入力データ

';**update**+books+set+title%3d'ECC'+where+id%3d'100' -- +

'';　　=　　=　　コメント化して無視

```
$sql = "select * from books where author = '" . $_GET['author'] . "'  order by id";
      ↓
$sql = "select * from books where author = '';update books set title='ECC'  where id='100' – order by id";      //生成されたSQL文（2つのSQ
```

;が入力されているのでいったんselect文は終了、order by idはコメント化され無視される。

その他の攻撃

OSコマンドの実行、ファイルの読み出し、ファイルの書き出し、HTTPリクエストにより他のサーバを攻撃

脆弱性の問題(リテラルの扱い)

リテラルの扱い方法により、攻撃されることがある。

1. 文字列リテラルの問題

   SQLの標準規格・・・文字列に'(シングルクォート)を含める場合、'(シングルクォート)を重ねる。

   <例>O'Reilly → O''Relliy

   <例>'(シングルクォート)を重ねるの忘れている場合、O'Relliyと入力。

```
$sql = “select * from books where author = ‘ . $_GET[’author’]. ’”;
       ↓
$sql = “select * from books where author = ‘O’Relliy’”;
                                          ‘O’・・・文字列（ここで終了）、Relliy'・・・はみ出した部分（構文エラー）→SQL文に変更＝SQLインジェクシ
                                           はみ出した部分・・・  ;delete from books
```

   <例>入力値　・・・　1';delete+from+books —

   ↓ 作成される文字列

   '1' ; delete from books—'

2. 数値リテラル

   PHPなどは型がない。

   <例>30';delete from employeeと入力した例

```
$sql = “select * from employee where age > “ . $_GET[’age’] . ”;
        ↓
$sql = “select * from employee where age > 30;delete from employee”;
        //
;は数値でないため以降がはみ出した部分になり実行されてしまう。
```

対策

プレースホルダ(?)にSQL文を作成する。

SQLを作成する時にリテラルを正しく構成するなどSQL文が変更されないようにする。

▼ プレースホルダによりSQL文の組み立て

？・・・プレースホルダ（場所取り）

<例>

```
$sql = “select * from books where author =  ?  order by id”;
```

フォームから入力されたデータは**？プレースホルダ**に格納される。

LIKE句

LIKE句では**ワイルドカード**として**_(任意の１文字)**や**%(０文字以上の任意の文字列)**が使用される。これらの文字がSQLインジェクションに利用されることがあるためエスケープ処理を行う必要がある。

<例>LIKE句の例

```
$sql = "select * from users where name like ‘%tanaka%’";   //nameに「tanaka」を含むSQL。
```

エスケープ対象文字　・・・　　_、%

エスケープしたい場合、escape関数で指定する。LIKE '%30-50!% off%' ESCAPE '!';

<例>

```
$sql = "select * from users where name like ‘%#%tanaka%’ESCAPE‘#’  ";        #に続く文字を通常のの文字として扱うことができる。この場合は「%tanaka
```

プレースホルダを用いた処理例

検索条件が動的に変わる　・・・　?を含んだSQL文を作成し、バインドする。

<例>PHPでのコーディング

```
$sql = “select * from users where id = ? ”;                                  //プレースフホルダの指定
$stmt = mysqli_prepare($conn, $sql);                //$connはデータベースのコネクション
mysqli_stmt_bind_param($stmt , “i”, $id);                                        //$idはintegerのためiを指定、バインドする
mysqli_stmt_execute($stmt)                                                                                                  //SQL文の実行
```

様々な列でのソート(order by句)

データベースのソート処理を行う時、ソートするための項目を入力することがある。そのような場合SQLインジェクションを利用されることがある。

```
$sql = “select * from users order by ” . $row;    //$rowはextractivalue(0,select+concat(’$’,id,’:’,pwd)+from+users+limit+0,1))
```

＊SQLインジェクションの保険的対策

詳細なエラーメッセージを抑止(PHPのデフォルトは無効)　・・・ **display_error = off**

入力値の妥当性を検証

データベースの最小限の権限を設定　・・・　grant文で設定

＊エスケープすべき文字の詳細　文字エンコーディングの影響