SQLインジェクションはWebアプリだけで発生するとは限りません

2016/01/08
Bing Webマスターツールの指摘に従っていくつか修正。ついでに文面も修正。

2008/05/04
Webアプリが外部に公開されているから事例が多く発生して見えるだけです。 人知れず自分の担当する「Webアプリではないシステム」でも起きている可能性が大きいのです。

だらだら書いてます。文章構成とか気にしてないので読みにくいかも。

Webアプリが外部に公開されているからSQLインジェクション事例が多く発生しているようにみえるだけ。

SQLインジェクションの例

マクロで発生

あるBIプロダクト SQL中に外部からの入力を埋め込むマクロが備わっています。

使い方は

```
SELECT *
   FROM PROD_MASTER
 WHERE PROD_CATEGORY = '1'
 #PromptMacro(value1,"AND PROD_CD=")#
```

value1 に xxx を設定すると

```
SELECT *
 FROM PROD_MASTER
WHERE PROD_CATEGORY = '1'
  AND PROD_CD=xxx
```

のような感じにSQLの一部が置換されます。

value1に値がまだ設定されていない場合は、プロンプトが表示され値の入力を促します。設定されると、以降他のvalue1を使うマクロは入力された値を適用してSQLを生成します。 よくある機能ですね。

ところでこれvalue1の入力値が検証できないと非常に危険です。マクロでは入力値の妥当性チェックが出来ないからxxxが

```
   "1 OR PROD_CD=2"
```

だったりすると

```
SELECT *
  FROM PROD_MASTER
 WHERE PROD_CATEGORY = '1'
   AND PROD_CD=1 OR PROD_CD=2
```

あらら。 もし項目名 PROD_CD が推測できちゃったら複数条件を指定できちゃいますね。
全部のレコードを抽出したいなら、項目名を知る必要はないです□xxxが

```
  "1 OR 1=1"
```

だったりすると

```
SELECT *
  FROM PROD_MASTER
 WHERE PROD_CATEGORY = '1'
   AND PROD_CD=1 OR 1=1
```

1=1 の式の結果は真でしょうから PROD_CD=1 以外のレコードも抽出対象になってしまいますね。

複数のクエリを注入することもできます□ value1 に

```
  "1 ; DELETE FROM PROD_MASTER"
```

を設定すると

```
SELECT *
  FROM PROD_MASTER
 WHERE PROD_CATEGORY = '1'
   AND PROD_CD=1 ; DELETE FROM PROD_MASTER

↓ SQLが2つ組み込まれている

SELECT * FROM PROD_MASTER WHERE PROD_CATEGORY = '1' AND PROD_CD=1
DELETE FROM PROD_DATA
```

これはいわゆるSQLインジェクションというやつです□SQLもしくはSQLの一部を注入(インジェクション)するわけですね。

入力したCSVファイルで発生

安直なWeb上のサンプルでコードを書くと遭遇する例。 以下に仮想的なコードを示します。

```
//// import data
var rdbObj = new rdbObject("id=xxx;pass=yyy;db=zzzz");
var fileObj = new fileObject("D:\\data.csv","r");
var sql = "";
var data1,data2;

while(fileObj.endOfFile != true) {
```

```
   fileObj.read(data1,data2);
   sql = "INSERT INTO PROD_DATA(ITEM_CD,QTY) VALUES('" + data1 + "'," +
data2 + ");";
   rdbObj.execute( sql );
 }

 rdbObj.close();
 fileObj.close();
```

data.csv から読み込んだ値を、 データベースの PROD_DATA テーブルへ INSERT しています。 もし□data.csv ファイルの 1 行目が

```
  "ABC","100"
```

であれば、

```
  INSERT INTO PROD_DATA(ITEM_CD,QTY) VALUES('ABC',100);
```

のSQLが作られ、実行されます□ data.csv ファイルの各行毎に上記のようなSQLが作られ、実行されていきます。 よくありがちな例です。

さて。
もし data.csv ファイルの 1 行目が

```
   "',100); DELETE FROM PROD_DATA; --","200"
```

だとどうなるでしょう？

```
  INSERT INTO PROD_DATA(ITEM_CD,QTY) VALUES('',100); DELETE FROM PROD_DATA;
--',200);

 ↓ SQLが2つとコメント1つが組み込まれている

  INSERT INTO PROD_DATA(ITEM_CD,QTY) VALUES('',100)
  DELETE FROM PROD_DATA
  --',200)
```

これもSQLインジェクションです。

SQLを動的に組み立てる危険性

SQLを文字列から動的に組み立てる処理は危険を抱えてます。

外部の信用ならないデータを取り込む

プログラムの内部で、プログラム作成者の意図に従って生成したデータ（文字列）で構成するSQLはある程度信用できるSQLです。 少なくとも意図的に妨害や損傷を与えるSQLを作る事はしない筈です。

ですが、ファイルやデータベーステーブルやユーザの入力等、プログラムの外部から取り込む、予期しない内容を入力してくる可能性がある場合、これを信用できないデータ、もしくは汚染されたデータと呼びます。
みんな良い人ならいいのですが、そうでない人もいるので、意図的に誤動作を誘発するようなデータ内容を送ってくる可能性があります。 外部プログラムも、もしかしたら悪意を持った内容を送り込んでくるかもしれません。

編集したSQL自体の信用検証

当然、入力値はきちんと検証を行い、プログラムの仕様に合ったものなのかを確認します。もちろん仕様を外れていればエラーにしなくてはいけません。

でも、仕様に合うのに問題がおきる場合もあります。
特にSQLを文字列の連結で作るような場合、入力値内に

```
' シングルコーテーション
" ダブルコーテーション
-- コメント開始文字
; マルチステートメント指示
```

のような文字がある場合はエスケープ処理を行い文字列中に含ませても問題がないよう対処しなければいけません。

あれ□SQLを実行したいだけなんだけどな……

入力値のせいでSQLが成り立たなくなるのはまずいですから、単にエスケープ処理で終わる訳ではなく、全部連結した結果SQLとして正しい体裁を為すように補正する場合もあります。

……おかしいですよね。 入力値そのものの検証ではなく、それを組み入れたSQLも検証しなくてはいけないなんて。 入力値の検証にコストをかけて、かつSQLの構成確認にコストをかけて。

サニタイズ？何それおいしいの？

世の中には「サニタイズ」なる用語があります。この用語を使っている書籍やページが言うには「 特殊文字はちゃんとサニタイズ(？)して無力化しておけば問題ないぜ」と。

別にやってもいいんですけどね。 苦労の割には本来の処理にあまり貢献できません。 これは「サニタイズ」が小細工の部類であり□SQLを文字列の連結で組み立てていく際に起こる問題を全く解決できないからです。

プレースホルダを使いましょう

そもそも、入力値を含めて作成された文字列をSQLとして解析しようとするから問題が起きます。

だったら□SQLが入力値の内容に左右されない様にしちゃえば良いわけです。
そのやり方の一つにプレースホルダという仕掛けがあります□RDBに限らず、こういうやり方はいろいろなところに使われています。 昨今のWebアプリのSQLインジェクションは、そういう機能を知らずに(教わらずに)きた者によって引き起こされていると思ってもいいのではないかと思います。 まぁ、教えられる者も居なくなってきているという話もあるのですが。

プレースホルダの例

以下に仮想的なコードを示します。

```
//// import data
var rdbObj = new rdbObject("id=xxx;pass=yyy;db=zzzz");
var fileObj = new fileObject("D:\data.csv","r");
var sql = "";
var data1,data2;

rdbObj.prepare("INSERT INTO PROD_DATA(ITEM_CD,QTY) VALUES(@itemcd,
@qty);");

while(fileObj.endOfFile != true) {
  fileObj.read(data1,data2);
  rdbObj.parameter("@itemcd", data1);
  rdbObj.parameter("@qty", data2);
  rdbObj.execute();
}

rdbObj.close();
fileObj.close();
```

最初に、実行したいSQLとそのSQLでつかうパラメタを定義しコンパイルします□prepare処理と言われる前処理です。
次にパラメタに値を割り当ててからSQLを実行します□ SQLはコンパイルされているので、毎回prepare処理を行う必要はなく、パラメタに割り当てる値を更新すれば次のデータを処理(SQLを実行)できることになります。

この□SQL中にある“@xxx”のようなパラメタのことをプレースホルダといいます。いい加減な説明なのであとでちゃんと調べたほうが良いですよ。

SQLServerだと“@xxx”□Oracleだと“:xxx”かな□PostgreSQLとかは？を使いますね。このときは名称ではなく□番目のパラメタ、として指定します。

すでにSQLはコンパイルされており、入力値によってSQLが改変されることはありません。入力値の特定文字のエスケープ処理も要りません。素直に値を設定すればよいのです。

でも油断はするなよ

もしデータベースの内容をWebブラウザへ出力するなら、出力中に怪しげなコードが含まれていてもそれを無効化するような仕掛けを入れます。 例えば、無事データベースに登録された文字列が

```
   □□コンパイラ用ライブラ
リ<JavaScript>document.href="http://xxxxx";</JavaScript>
```

だったりしたら、この内容をそのままブラウザに出力するとそのブラウザに別サイトへのアクセスを指示することができちゃうかもしれません。

やっとここで " サニタイズ " の出番です。
文字列中の " < " や " > " をそれぞれ"<"や " >"のような実体参照表記に変更するなどの処理を実施

します。

でもたいていの場合、専用の関数やメソッドが用意されていて自分でサニタイズ処理を行うような関数やメソッドを作らなくてもいい事がほとんどです。
というか作らない方がいいです。間違いなく考慮漏れや抜けが発生しますから。

database, SQL, SQLインジェクション, プレースホルダ, 技術資料