SQLインジェクション対策再考

2008/07/05
HASHコンサルティング株式会社
代表取締役 徳丸 浩
http://www.hash-c.co.jp/

アジェンダ

- 本日の構成
  - 正しくないSQLインジェクション対策の今昔
  - SQLインジェクション対策の考え方
  - SQLインジェクション対策の実際
- 議論の焦点
  - 入力値検証とは何か
  - SQLのエスケープ方法詳細
  - 数値項目の対策
  - 最近のトピックス

第1部

正しくないSQLインジェクション対策の今昔

正しくない例1(2001年)

一つの方法として，文字列中に「'」が登場した場合，「''」と重複してやることで，文字列を囲むシングル・クォーテーションではなく，文字としてのシングル・クォーテーションとして評価されるようにするというやり方がある。

【中略】

特にセキュリティを重視した場合においては，Webアプリケーションの仕様上許されるのであれば，「'」以外にも，半角の特殊文字列やSQL 文の予約文字列（SELECTやINSERTなど）を全角にしてからデータベースに格納するというようなチェック機構を作っておくことが望ましい。

正しい

これは余計

でもこれは
2001年の記事だから…

実践! セキュアなWebプログラミング　日経オープンシステム2001年5月号 から引用

正しくない例2(2007年)

(1) **入力値のチェック**
【中略】
データベースで扱う値に対して上記のような文字種、文字数等の条件を明確にし、ブラウザから渡された値が、入力値として正しい形式であるかどうかをチェックする。条件を[illegible]厳密にチェックすることによって、任意のSQL文の混入を避けることができる。

場合もある

(2) **特殊記号のエスケープ**
1) **シングルクォート「'」のエスケープ**

正しい

【中略】
3) **セミコロン「;」の拒否**
次の特殊記号が含まれているときは、パラメータを受理しない

わけにいかない

; → **受理しない**

「;」は、SQL文のコマンドの区切りに使用される。
【中略】
5) **その他の特殊記号のエスケープ**（Microsoft Jetエンジン）
またMicrosoftのJetエンジンでは、次の文字も機能をもつ特殊記号として扱われる。

| VBAステートメント実行文字

どうやってエスケープ?

IPA ISEC **セキュア・プログラミング講座**：Webアプリケーション編 **第6章 入力対策**：SQL注入 **より引用**
http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/502.html

正しくない例3(2008年)

【前提1】
・Webアプリケーションは文字列を入力として受理できる
・リレーショナル・データベース管理システムと連携

【入力の例】
DECLARE%20@S%20NVARCHAR(4000)SET%20@S=hogehoge EXEC(@S)

パーセントエンコードをデコードしてからでないと無意味

・文字列として入力
・特殊文字を文字として扱うために「¥」を挿入
・「;」は削除

セミコロンを勝手に削除しないで!

むやみに「¥」を挿入しても…

【入力値チェックの結果】
DECLARE¥%20@S¥%20NVARCHAR¥(4000¥)SET¥%20@S¥=hogehoge EXEC¥(@S¥)

【前提2】
・SQLクエリーはアプリケーションで生成
・SQL構文に用いるような文字列はユーザーの入力としてはありえない

意味不明

【入力の例(入力値チェックの結果)】
DECLARE¥%20@S¥%20NVARCHAR¥(4000¥)SET¥%20@S¥=hogehoge EXEC¥(@S¥)

・SQL構文に用いられる代表的な文字列をフィルタリングして削除
・アットマーク(@)はデータベース上で変数の識別子やスクリプトの実行に用いられることがあるため削除

サニタイズ!!

【サニタイジングの例】
¥%20S¥%20¥(4000¥) ¥%20S¥=hogehoge ¥(S¥)

被害が続くSQLインジェクション攻撃，もう一度対策を見直そう より引用
http://itpro.nikkeibp.co.jp/article/COLUMN/20080514/301660/

なぜ誤った解説がなくならないのか

- 攻撃方法からの発想
  - 攻撃に使用する文字･文字列を削除･改変するアプローチ
  - いわゆる「サニタイズ」
  - 脆弱性が混入する根本原因からのアプローチではない
- 実はアプリケーションなんか書いたことがない人が説明している?
  - セキュリティのプロが全員アプリケーションを書けるとは限らない
- そのサンプルコード、動かしてみた?
  - でもテスト環境構築するだけでも大変だしぃ
- コピペの悪弊
  - 昔の間違った解説が延命されられる

みんな攻撃が大好きだww

【参考】なぜセミコロン「;」を削除したがるのか?

- **実はセミコロンの削除には実効的な意味はあまりない**
- **セミコロン削除の意図は、複文実行の防止と思われる**
  - SELECT * FROM XXX WERE ID=''**;**UPDATE XXX SET …
- SQL**インジェクションの文脈で複文が実行できるのは、**MS SQL とPostgreSQL
- **現実に**MS SQL**は、複文を使った改ざん事件が多発**
- **しかし、**MS SQL**は、セミコロンなしでも複文が書ける**
  - SELECT * FROM XXX WERE ID=''UPDATE XXX SET … **でもよい**
- **すなわち、セミコロンの削除で保険的にせよ意味があるのは、**PostgreSQL**の場合だけ**

続きはWebで http://www.tokumaru.org/d/20080502.html

http://www.tokumaru.org/d/20080627.html

「危険文字と言わないで」

2007-09-11 [ネタ][セキュリティ]メタ文字たちが騒いでいます 編集

■ 危険文字と言わないで 12:57 tB 22 users ★★★★★★★★★★★★★★★

昨日、とあるベンダーのセキュリティセミナーに行きました。

そしたら、どうです。僕らのことを「危険文字」と呼んでる人がいるじゃありませんか。

ひどすぎます。僕らは日夜、世界中で、HTMLのページを書いたり、データベースを検索したりで、Webを支えているんです。僕らが無かったら、Yahoo!だって、Amazonだって、Googleだって、tokumaru.orgだって実現できないんですよ。うそだと思ったら、「<」も「>」も「'」も「"」もなしに、Webアプリを書いてごらんなさい。

こんなことを言ったら、UTF-7なら、7bitアスキーなら「危険文字」使わないで書けるとか言う人がいるかもしれない。でも考えてみてください。UTF-7とか7bitアスキーの方こそ、よっぽど「危険」じゃないですか。それに、UTF-7は見た目が変なだけで(UTF-7怒るかな?)、結局は僕らのことを表現しているんです。当たり前じゃないですか。

だから今後は、危険文字どころか、「有用文字」と呼んでもらいたいですね。

今度僕らのことを「危険文字」なんて言ったら、その人には僕たちもう使わせてあげない。

わかりましたか?

ブンブンブン

SQLインジェクション対策の考え方

そもそもなぜSQLインジェクションが発生するのか?

- 原因は、リテラルとして指定したパラメータが、**リテラルの枠をはみ出し**、SQLの一部として解釈されること
- 文字列リテラルの場合
  - シングルクォートで囲まれた(クォートされた)範囲をはみ出す

```
SELECT * FROM XXX WHERE A=''OR'A'=A'
```

- 数値リテラルの場合

はみ出した部分

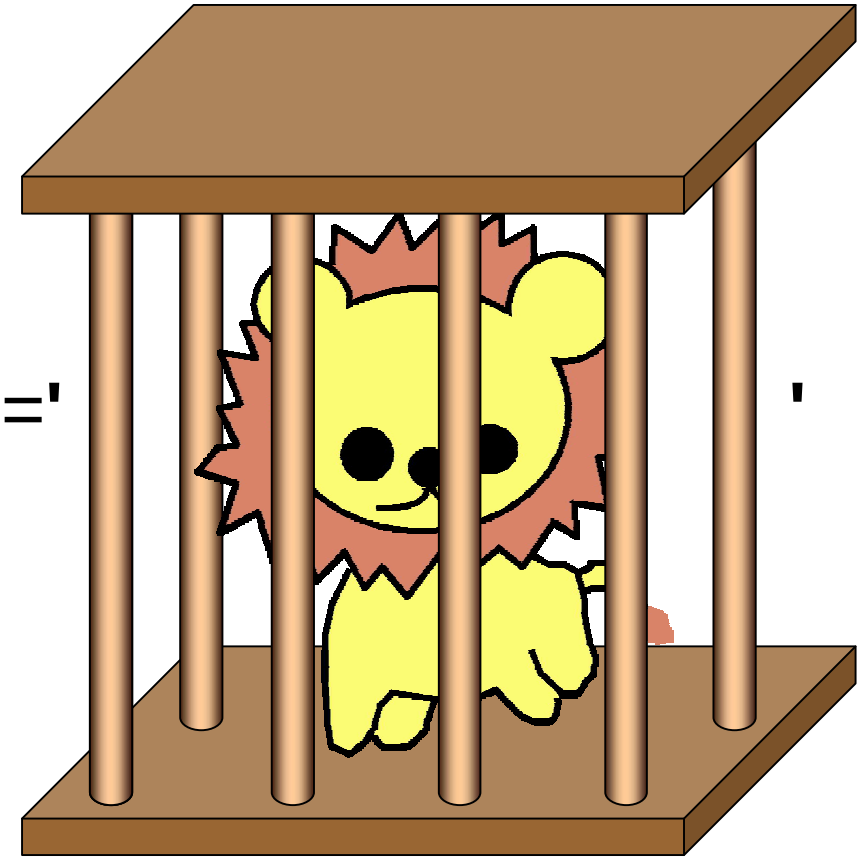
  - 数値でない文字(空白、英字、記号など)を使う

```
SELECT * FROM XXX WHERE A=1OR TRUE
```

エスケープは檻にしっかり入れるイメージ

- 檻に入っている分には、中身の「危険性」を気にする必要なはい
- 危険性がなくても、檻から出てしまうのはバグ

select * from animals whre kind='



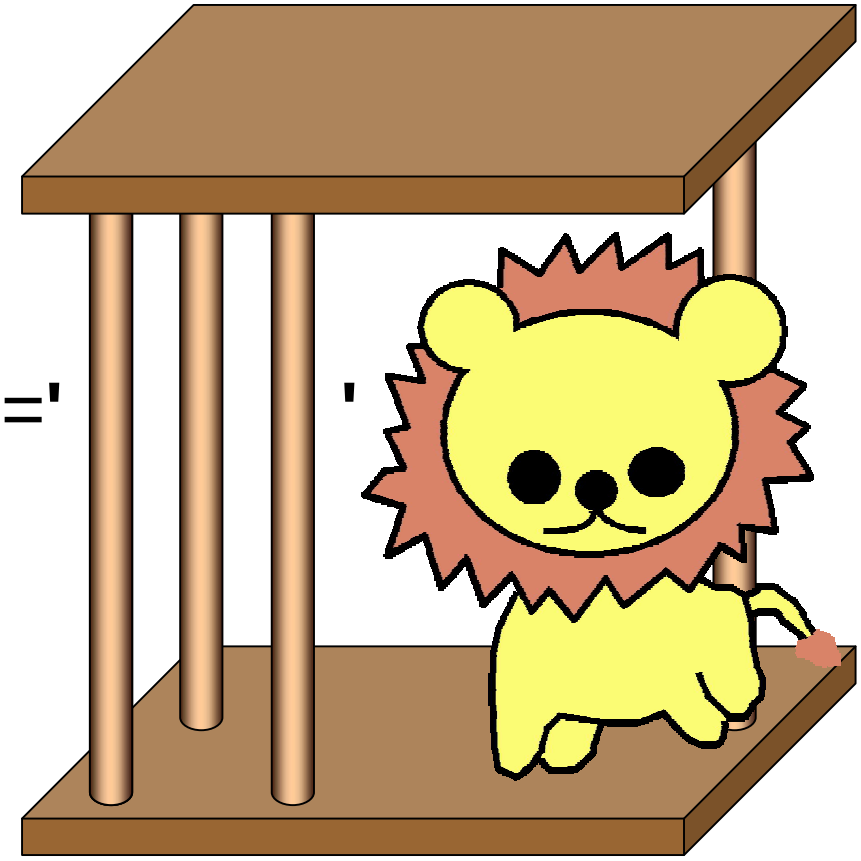
'

「危険な文字・文字列」; | ' @ declare union xp_cmdshell @ ...

SQLインジェクションは檻から逃げるイメージ

- **パラメタがリテラルからはみ出し、SQL文の命令として解釈される状態**

select * from animals whre kind='



'

「危険な文字・文字列」; | ' @ declare union xp_cmdshell @ ...

[プロの礼儀作法としての参考文献] ライオン･エスケープ

http://www.rakuten.co.jp/torito/659325/660549/ **から引用**

SQLインジェクション対策の実際

ではどうすればいいのか?

- **基本は、リテラルをしっかりと檻に閉じ込めること**
  - **方法1:バインド機構の利用(推奨)**

```
my $sth = $db->prepare("SELECT ERRMSG FROM ERRINFO WHERE ERRNO=?");
my $rt = $sth->execute($n);
```

    - **バインド機構の実装にバグがない限り安心(どうやって確認する?)**

  - **方法2:SQLの動的組み立て+エスケープ**
    - **ただし数値の場合は別の方法が必要**

第二の選択肢　動的SQL生成+エスケープ

- なぜバインド機構を利用しないのか
  - プラットフォームの制約
  - フレームワークの制約
  - 古いアプリケーションの保守

動的SQL生成の場合は文字列と数値で対応が変わる

- 文字列の場合
  - エスケープ
- 数値の場合
  - 変数に型のある言語 Java、C#など
    - 数値型を使っている場合は問題ない
    - 文字列型を使って数値を処理する場合は、変数に型のない言語と同じ方法
  - 変数に型のない言語 Perl、PHP、Ruby、VBScript(ASP)など
    - 数値の妥当性確認

文字列リテラルのエスケープ

- どの文字をエスケープするのか?
  - SQL製品の文字列リテラルのルールに従う
  - ISO標準では、「'」→「''」
  - MySQLとPostgreSQLは「'」→「''」 「¥」→「¥¥」
  - PostgreSQLの場合は、standard_conforming_stringsおよびbackslash_quoteの影響を受ける
    - standard_conforming_strings=onの場合は、ISO標準と同じ方法になる
    - backslash_quote=off の場合は、「'」→「¥'」というエスケープがエラーになる

	元の文字	エスケープ後
Oracle MS SQL IBM DB2	’	’’
MySQL	’	’’ または ¥’ (’’を推奨)
PostgreSQL	¥	¥¥

【参考】商用RDBMSの文字列リテラルの定義

- Oracle:cは、データベース・キャラクタ・セットの任意の要素です。リテラル内の一重引用符 (') の前には、エスケープ文字を付ける必要があります。*リテラル内で一重引用符を表すには、一重引用符を2つ使用します*。

  http://otndnld.oracle.co.jp/document/products/oracle10g/102/doc_cd/server.102/B19201-02/sql_elements.html#41297

- DB2:ストリング区切り文字で始まりストリング区切り文字で終わる文字のシーケンス。この場合のストリング区切り文字はアポストロフィ (') です。【中略】*文字ストリング内で 1 つのストリング区切り文字を表したいときは、ストリング区切り文字を 2 つ連続して使用します*。

  http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=/com.ibm.db2.luw.sql.ref.doc/doc/r0000731.html

- MS SQL:*単一引用符で囲まれた文字列に単一引用符を埋め込む場合は、単一引用符を 2 つ続けて並べることで 1 つの単一引用符を表します*。

  http://technet.microsoft.com/ja-jp/library/ms179899.aspx

【参考】MySQLの文字列リテラルの定義

8.1.1. **文字列**

一部のシーケンスは、個々の文字列内で特別な意味を持ちます。これらのシーケンスは、いずれも、エスケープ文字として知られるバックスラッシュ（‘¥’）で始まります。MySQLでは、次のエスケープシーケンスが認識されます。

¥0	ASCII 0 (NUL) 文字。
¥'	単一引用符 (“'”) 文字。
¥"	二重引用符(“"”) 文字。
¥b	バックスペース文字。
¥n	改行文字(LF)。
¥r	復帰改行文字。
¥t	タブ文字。
¥Z	ASCII 26 (Control-Z)。表の下部にある注釈を参照してください。
¥¥	バックスラッシュ(‘¥’)文字。
¥%	‘%’文字。表の下部にある注釈を参照してください。
¥_	‘_’ 文字。表の下部にある注釈を参照してください。

¥' 単一引用符 (“'”) 文字。

文字列に引用符を含める方法は、いくつかあります。

- ***‘’’で囲んだ文字列内で、‘’’を使用する場合、‘’’’と記述することができます。*【後略】**

「MySQL :: MySQL 5.1 リファレンスマニュアル :: 8.1.1 文字列」から引用
http://dev.mysql.com/doc/refman/5.1/ja/string-syntax.html

【参考】PostgreSQLの文字列リテラルの定義

4.1.2.1. 文字列定数

SQLにおける文字列定数は、単一引用符 (') で括られた任意の文字の並びです。例えば、'This is a string'です。***文字列定数内の単一引用符の記述方法は、2つ続けて単一引用符を記述することです***。

【中略】

エスケープ文字列の中では、バックスラッシュ文字 (¥) によりC言語のようなバックスラッシュシーケンスが始まります。バックスラッシュと続く文字の組み合わせが特別なバイト値を表現します。¥bは後退 (バックスペース) を、¥fは改頁を、¥nは改行を、¥rは復帰 (キャリッジリターン) を、¥tはタブを表します。また、¥digitsという形式もサポートし、digitsは8進数バイト値を表します。¥xhexdigitsという形式では、hexdigitsは16進数バイト値を表します。(作成するバイトの並びがサーバの文字セット符号化方式として有効かどうかはコード作成者の責任です。) ここに示した以外のバックスラッシュの後の文字はそのまま解釈されます。***したがって、バックスラッシュ文字を含めるには、2つのバックスラッシュ (¥¥) を記述してください。また、通常の''という方法以外に、¥'と記述することで単一引用符をエスケープ文字列に含めることができます***。

http://www.postgresql.jp/document/current/html/sql-syntax-lexical.html#SQL-SYNTAX-CONSTANTS

MySQLとPostgreSQLで「¥」のエスケープが必要な理由

SELECT * FROM XXX WHERE ID='$id'

$id として ¥'or 1=1# が入力されると

¥'or 1=1#

↓ エスケープ(「¥」のエスケープをしない場合)

¥''or 1=1#

元のSQLに適用すると、

SELECT * FROM XXX WHERE ID='¥'' or 1=1#'

すなわち、SQLの構文が改変された(「¥’」で「’」一文字と見なされる)

Shift_JISの問題

DB側の日本語処理が不完全な場合

表		'
0x95	0x5c	0x27

↓フロント側でのエスケープ処理

表		'	'
0x95	0x5c	0x27	0x27

↓データベース側の解釈

0x95	0x5c	0x27	0x27
0x95	¥' で'一文字		' がエスケープされずに余る

フロント側の日本語処理が不完全な場合

表		'
0x95	0x5c	0x27

↓フロント側でのエスケープ処理(0x5cと0x27をそれぞれエスケープ)

0x95	0x5c	0x5c	0x27	0x27

↓データベース側の解釈

0x95	0x5c	0x5c	0x27	0x27
「表」一文字		¥'で一文字		' がエスケープされずに余る

文字コードの問題

- 例えば、Shift_JISを避ける
- 言語レベルで文字コードを意識したものを
  - Java　　# 元々内部はUnicode
  - Perl5.8　　# それ use utf8; で
- PostgreSQLの対応(8.1.4)

・常にサーバ側で無効なコードのマルチバイト文字を拒否するように修正されました (8.1系～7.3系)

これにより、一律にすべての文字エンコーディングのすべてのテキスト入力に対して検査が行われ、単に警告が出るのではなく常にエラーが出るようになりました。この変更は CVE-2006-2313 に記述されているような SQLインジェクション攻撃に対抗するものです。

・文字列リテラル中の安全でない「¥'」を拒否する機能が追加されました (8.1系～7.3系)

CVE-2006-2314 に記述されている類のSQLインジェクション攻撃をサーバ側で防ぐため、SQL 文字列リテラルとして ’’ だけを受け付け、¥’ を受け付けないように変更されました【中略】

この振る舞いを調整するため、新たな設定パラメータ backslash_quote が追加されました。なお、CVE-2006-2314 を完全に防ぐには、おそらくクライアント側の修正も必要です。
backslash_quote は、安全でないクライアントが「安全でない」ことを明らかにするのを目的としています。

http://www.sraoss.co.jp/PostgreSQL/8.1.4/changes.html

数値リテラルの場合

- 数値としての妥当性確認をSQL組み立て時に行うとよい
- 最初に妥当性検証していても気にしないで再度検証する
  - 大したオーバーヘッドにはならない

```
# 呼び出し例
eval {
    my $sql = "SELECT ERRMSG FROM ERRINFO WHERE ERRNO=" . int_check($n);
    ....
}
if ($@) {
    # エラー発生時の処理
}
...
# 整数チェック関数の例
sub int_check {
    my $val = shift;
    if ($val =~ /^-?[0-9]+$/) {
        return $val;
    } else {
        die "整数値エラー";  # エラーメッセージは$@に格納
    }
}
```

SQLエスケープの実装はどれを使う?

- 「安全なウェブサイトの作り方改定第3版(P7)」には、以下の記述があるが、
  - データベースエンジンによっては、専用のエスケープ処理を行うAPI を提供しているものがあります（たとえば、Perl ならDBIモジュールのquote()など）ので、それを利用することをお勧めします。
- DBIのquote()が全てDB側で用意したAPIを呼んでいるわけではない
  - DBD::PgPPでの実装

```
$s =~ s/(?=[¥¥¥'])/¥¥/g;        # PostgreSQLのAPIを呼んでいない
return "'$s'";
```

- 「安全なウェブサイトの作り方」の趣旨には同意するが、具体的にどの関数・メソッド・APIなら安全というガイドラインがないと、開発現場では使えない
  - プロジェクトの度にコンサルタント雇って調べさせる?

入力値検証は何をすればよいか

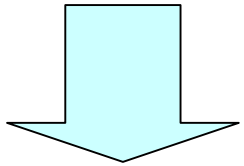
- アプリケーションレベルで、「'」や「;」を削除、あるいは拒否するわけにはいかない
  - クォートやセミコロンも正しく処理できるよう、バインド機構やエスケープを用いる
- ミドルウェアレベルでは、「半端なマルチバイトコード」や「UTF-8の冗長表現」をチェックし、エラーにすべき
- アプリケーションレベルでは、「業務要件」にしたがって入力値検証する
  - 結果としてSQLインジェクション対策になる場合もあれば、ならない場合もある
  - メタ文字ではなく、制御文字(ヌル文字を含む)のチェックは必要
    (改行・タブ以外の制御文字はミドルウェアレベルでチェックして欲しい)

おまけ:述語LIKEのワイルドカードのエスケープ

- 述語LIKEのワイルドカード「_」、「%」にも注意
- 狭義のSQLインジェクションとは違うが、サーバーに過負荷をかける場合がある(全件検索)
- MySQL以外の場合はESCAPE句の利用
  - … LIKE '#%%' ESCAPE '#'　-- 「#%」で文字としての「%」を示す
- MySQLの場合は「¥」によりエスケープ
  - … LIKE '¥%%'　-- 「¥%」で文字としての「%」を示す
- 商用RDBMSの場合は全角の「＿」や「%」にも注意

まとめ･提言

- そろそろ「入力値の未検証」という表現はやめよう
- バインド機構の利用促進
- 正しいエスケープ方法の普及
- 安全なフレームワーク
- 安全なバインド機構はどれ?
- 安全なエスケープ関数はどれ?



- 「安全なウェブサイトの作り方」のさらに具体的なガイドラインの必要性

ご清聴ありがとうございました