2.1.3 Avoiding SQL injection

According to folklore, one may avoid SQL Injection using *stored procedures*. That is not entirely correct. Stored procedures are named pieces of code installed on the database server. Rather than passing a full SQL query, the client will invoke the stored procedure by referencing its name, passing any parameters as needed. Quite like calling a programming language function or procedure.

An example: On an MS SQL Server, a stored procedure for inserting a name and an age in a `person` table might look like this:

```
CREATE PROCEDURE insert_person
                 @name VARCHAR(10),@age INTEGER AS
  INSERT INTO person (name,age) VALUES (@name,@age)
GO
```

The procedure, named `insert_person`, accepts a string parameter representing the name, and an integer parameter containing the age. A traditional `INSERT` statement is performed to update the table. The good thing is that `@name` and `@age` are typed variables, so we shouldn't encapsulate the name in quotes inside the procedure.

Problems may occur depending on how the procedure is called. If we do it really simply, we build an SQL query string that calls the procedure. The

following code accepts a `name` and `age` parameter from a GET or POST request, and calls the stored procedure with those values. As we have the false impression that a program using stored procedures is immune to SQL Injection, we don't do any metacharacter handling:

```
conn.Execute("insert_person '" & Request("name") _
                              & "'," & Request("age"))
```

Let's say an attacker enters a `name` that looks like the following, and keeps the `age` empty:

```
bar',1 DELETE FROM person --
```

Due to the simple string concatenation above, our program will ask the database to execute:

```
insert_person 'bar',1 DELETE FROM person --',
```

First a quite normal call of the stored procedure, then a `DELETE` statement that removes everything from the table. The stored procedure didn't help a bit, unfortunately.

It's about access rights in the database, some people say. We'll have to restrict the database user so that it is only allowed to execute stored procedures. No `SELECT`, no `INSERT`, no `DELETE`, no nothing. Partly right again, but only partly.

First, those restrictions would have prevented the above attack, but they wouldn't prevent the attacker from calling another stored procedure if he knew about it: the query would still be vulnerable to SQL Injection. That other stored procedure could perform the deletion for him if it was written to do so. And second, as a programmer I feel a little uncomfortable with leaving the security of my program to the settings of a database server. Database settings are often outside the control of the programmer. And you never know what those settings will be after an upgrade, a restore from backup, or just twiddling by an inexperienced admin trying to fix some problem. Database permissions should be utilized, but only as a secondary measure. At least as long as the problem we're trying to solve is SQL Injection.