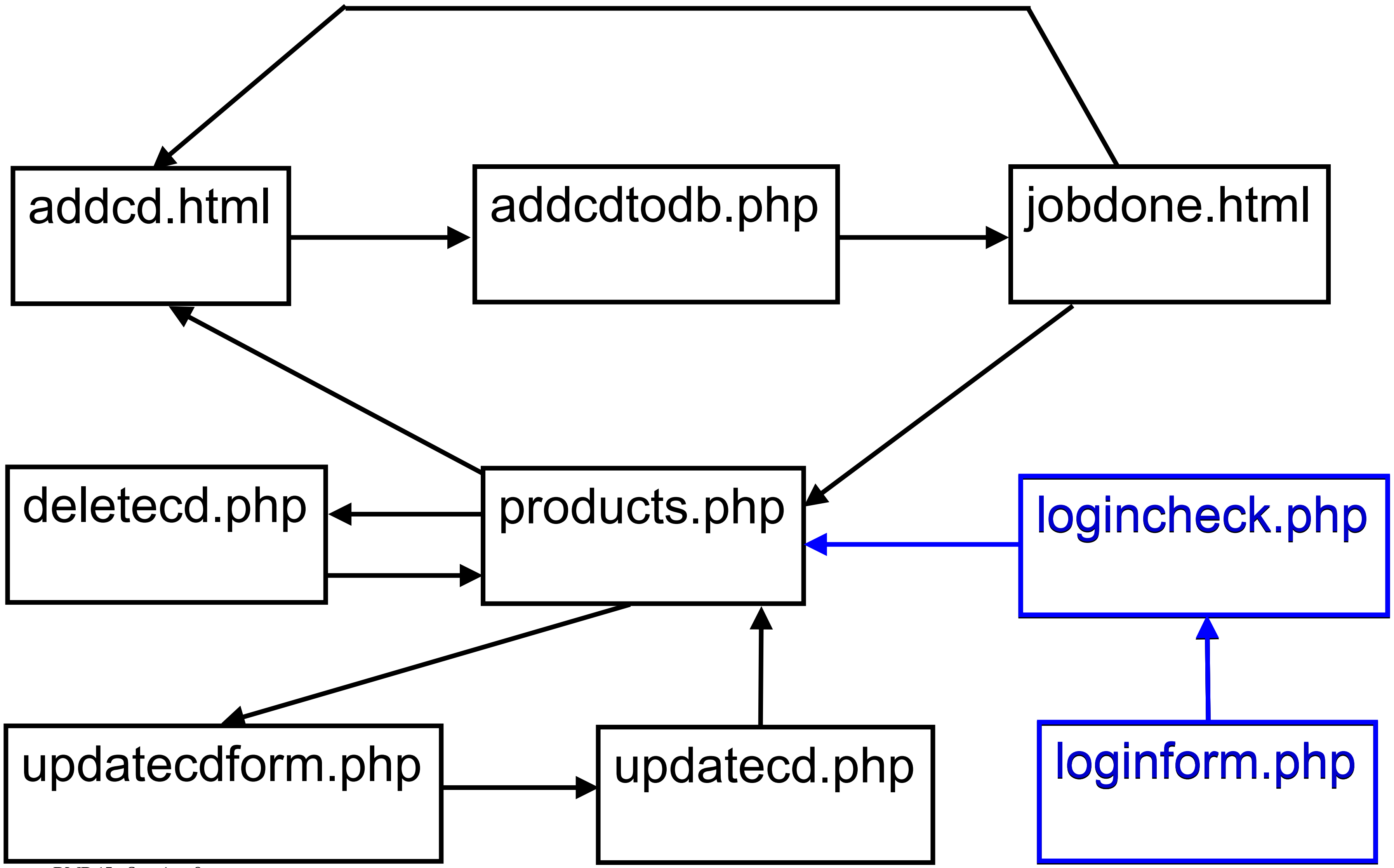
Dynamic Web Development

Lecture 15 – Security

Note - most of the examples use the old PHP mysql library, rather than the newer mysqli library

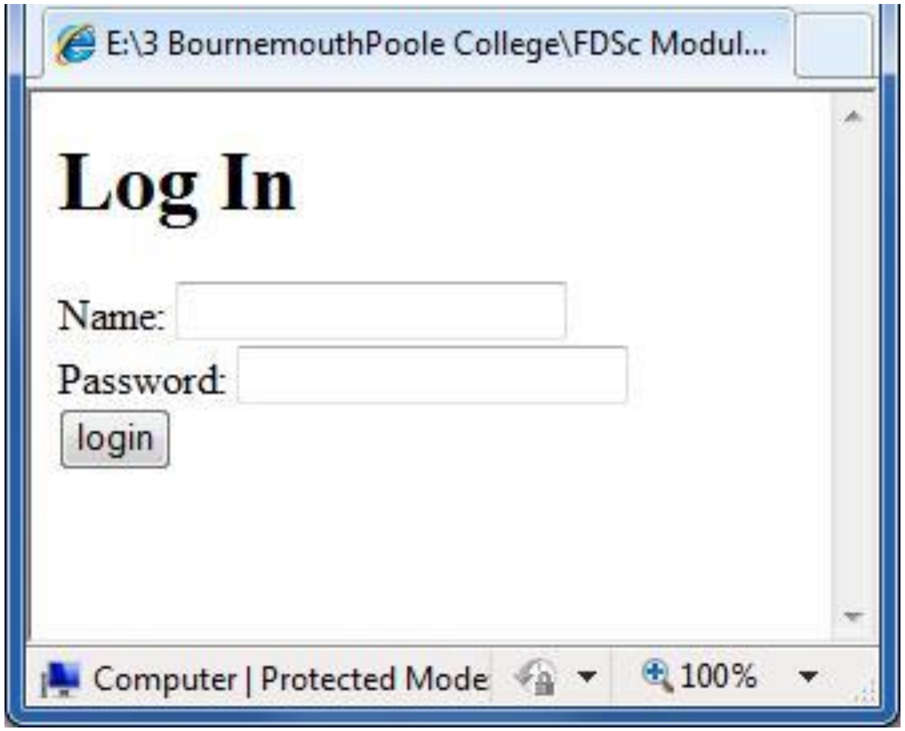
Page Map for updating a cd



loginform.php

```
<html>
  <head></head>
  <body>
   <h1>Log In</h1>
   <form name="login" action="logincheck.php"
                                  method="post">
          Name:
      <input type="text" name="username"></input>
      <br />
      Password:
      <input type="password" name="password"></input>
      <br />
      <input type="submit" name="submit"
                          value="login"></input>
    </form>
  </body>
</html>
```

loginform.php


E:\3 BournemouthPoole College\FDSc Modul...
Log In
Name:
Password:
login
Computer | Protected Mode
100%

```php
<?php
require "dbconn.php";

$username = $_POST['username'];
$password = $_POST['password'];

$sql = "SELECT * FROM person WHERE name ='".$username.
                                  "' AND password='".$password."'";

$result = mysql_query($sql) or die(mysql_error());

$row = mysql_fetch_array( $result );

if ($row != null)
  {
   header("Location: products.php");
   exit();
  }
else
  {
   header("Location: loginform.php?showerror=1");
   exit();
  }
?>
```

Note that I have included

showerror=1

into the line that returns the user back to the login form?

What might I do with that?

Using Sessions for Security

If someone types in:

http://www.kev.com/products.php

they can bypass the login page and get into your website.

We will use a session variable to control access to our pages.

session_start();

To start a PHP session, we must use the session_start() function.

This create a session if none exists, or maintain a session if one has already been started.

The function must be the very first line after the opening <?php tag.

There must be no blank lines, or other PHP statements before it.

Eg

```
<?php
session_start();
```

We can now set a session variable that tracks whether a user is logged in.

We will use this to redirect any user attempting to access pages directly to the **loginform.php** script.

We need to modify the logincheck.php script as follows:

```php
<?php
session_start();
require "dbconn.php";

$username = $_POST['username'];
$password = $_POST['password'];

$sql = "SELECT * FROM person WHERE name ='".$username.
                         "' AND password='".$password."'";

$result = mysql_query($sql) or die(mysql_error());

$row = mysql_fetch_array( $result );

if ($row != null)
  {
   $_SESSION['username'] = $row['name'];
   header("Location: products.php");
   exit();
  }
else
  {
   header("Location: loginform.php?showerror=1");
   exit();
  }
?>
```

How to protect the other pages

At the top of all other pages to which we wish to restrict access, we can add the following check:

```
<?php
session_start();
if ( !isset($_SESSION['username']) )
{
header("Location:loginform.php");
exit();
}
...
```

User Feedback

We can use session variables to personalise the pages. For example:

```
Welcome to your page  <?php echo $_SESSION['username'] ?>
```

Would display the following message on the webpage:

```
Welcome to your page Kevin
```

Logging Out

It is best to include a way for the user to explicitly log out.

Include a link on the products page

```
<a href="logout.php">click here to logout</a>
```

which takes the user to the following script:

logout.php

```
<?php
session_start();
session_destroy();
header("Location:loginform.php");
exit();
?>
```

How to Protect a Website

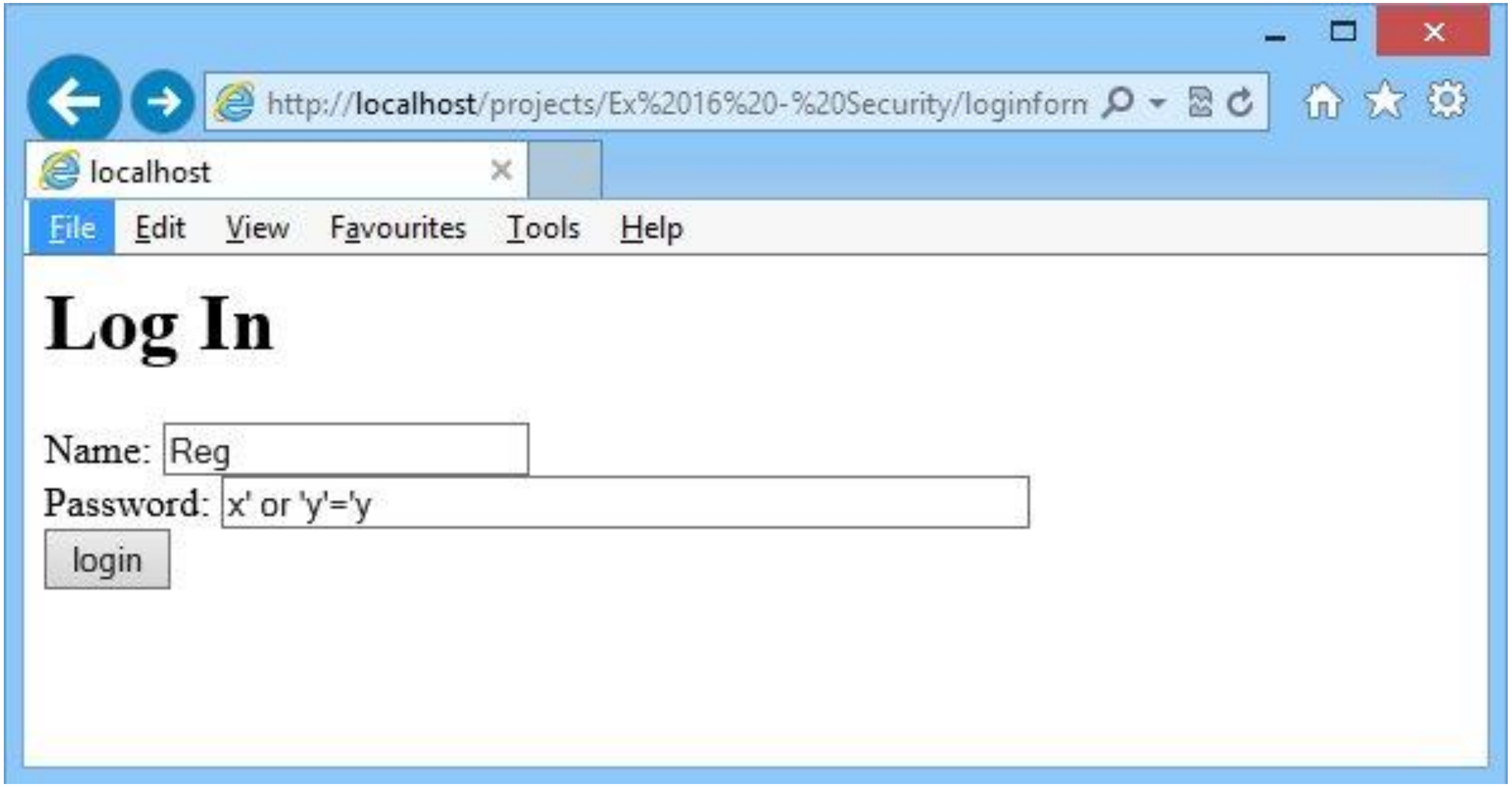
Person table

The database contains a table called person which was created with the following code:

```
CREATE TABLE person
(
name   VARCHAR(30),
password VARCHAR(30),
PRIMARY KEY (name)
);

INSERT INTO person  VALUES ('kevin', 'pass123');
INSERT INTO person  VALUES ('helen', 'pass789');
```

When I Type in This:

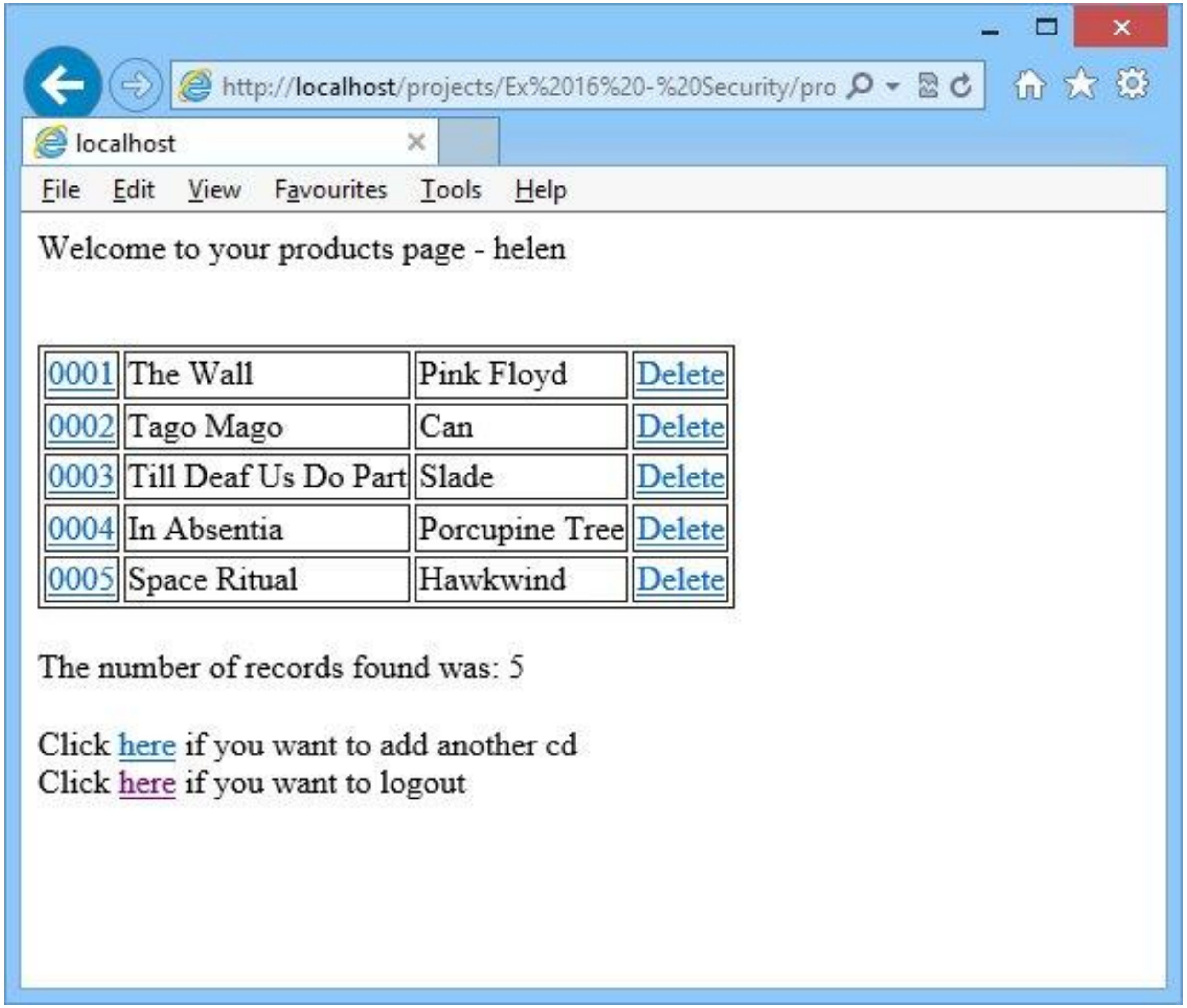


Log In

Name: Reg

Password: x' or 'y'='y

It Takes Me Straight Here:


http://localhost/projects/Ex%2016%20-%20Security/pro
localhost
File Edit View Favourites Tools Help

Welcome to your products page - helen

0001	The Wall	Pink Floyd	Delete
0002	Tago Mago	Can	Delete
0003	Till Deaf Us Do Part	Slade	Delete
0004	In Absentia	Porcupine Tree	Delete
0005	Space Ritual	Hawkwind	Delete

The number of records found was: 5

Click here if you want to add another cd
Click here if you want to logout

What Happened?

```
$username = $_POST['username'];
$password = $_POST['password'];

$sql = "SELECT * FROM person WHERE name ='".$username.
                         "' AND password='".$password."'";
```

When we substitute the values into the SQL string we get:

```
SELECT * FROM person
WHERE name = 'Reg' AND password = 'x' or 'y'='y'
```

which evaluates to:

```
SELECT * FROM person
WHERE FALSE AND FALSE or TRUE
```

which evaluates to:

```
SELECT * FROM person
WHERE FALSE or TRUE
```

which evaluates to:

```
SELECT * FROM person
WHERE TRUE
```

which will retrieve every record from the person table into $result.

```
$result = mysql_query($sql) or die(mysql_error());
```

The next line will get the first record from $result, which in this case happened to be the one for 'helen'.

```
$row = mysql_fetch_array( $result );

if ($row != null)
  {
   $_SESSION['username'] = $row['name'];
   header("Location: products.php");
   exit();
  }
```

This is SQL Injection.

Creating or altering existing SQL commands in order to expose hidden data.

..or worse.

Now before we go any further, let's change subjects for a while.

Computer Misuse Act 1990

1 Unauthorised access to computer material.

(1) A person is guilty of an offence if—

(a) he causes a computer to perform any function <u>with intent</u> to secure access to any program or data held in any computer , or to enable any such access to be secured;

(b) the access <u>he intends</u> to secure, or to enable to be secured, is unauthorised; and

(c) he knows at the time when he causes the computer to perform the function that that is the case.

(2) The intent a person has to have to commit an offence under this section need not be directed at—

(a) any particular program or data;

(b) a program or data of any particular kind; or

(c) a program or data held in any particular computer.

http://www.legislation.gov.uk/ukpga/1990/18

Another Example

```
$query = "UPDATE usertable
       SET password = '".$pwd."'
       WHERE name = '".$name."';";
```

so what happens if someone types in the following:

```
password:   mypass

name:    ' or name like '%admin%

the query becomes

UPDATE usertable
   SET password = 'mypass'
   WHERE name = '' or name like '%admin% ';
```

Any name containing the string admin has its password changed.

Or

```
SELECT * FROM person
WHERE name = 'Reg'
AND password = 'x'; DROP TABLE users; --'
```

The -- symbol is for SQL comments.

It causes SQL to ignore the final quote.

You don't know for sure that there is a table called users, but it's not a bad first guess.

Note:

The PHP mysql library does not allow execution of 2 queries in one statement.

Other libraries or languages may not be as robust.

The more about your website an attacker knows, the more vulnerable you are.

Attacking the database host machine's operating system (MSSQL Server)

```
$query  = "SELECT * FROM products WHERE id LIKE '%$prod%'";
   $result = mssql_query($query);
```

If attacker submits the value

```
 a%' exec master..xp_cmdshell 'net user test testpass /ADD' --
```

to *$prod*, then the *$query* will be:

```
$query  = "SELECT * FROM products
              WHERE id LIKE '%a%'
              exec master..xp_cmdshell 'net user test testpass /ADD' --%'";
   $result = mssql_query($query);
```

MSSQL Server executes the SQL statements in the batch including a command to add a new user to the local accounts database.

If this application were running as ***sa*** and the MSSQLSERVER service is running with sufficient privileges, the attacker would now have an account with which to access this machine.

Avoidance Techniques

- While it remains obvious that an attacker must possess at least some knowledge of the database architecture in order to conduct a successful attack, obtaining this information is often very simple.
- For example, if the database is part of an open source or other publicly-available software package with a default installation, this information is completely open and available.
- This information may also be divulged by closed-source code - even if it's encoded, obfuscated, or compiled. Don't make do with the default settings.
- Your own code may give information to the attacker through the display of error messages. Make sure that there are no errors before the code is transferred from the development server to the production server (preferably via the test server)
- Other methods include the user of common table and column names. For example, a website that links to a 'users' table with column names like 'id', 'username', and 'password' isn't a good idea.

dbconn.php

So what is the problem with this code, from a security point of view?

```
<?php
$host = "localhost";
$user = "root";
$password = "";
$database = "securitytest";

//connect to MySQL

$connect = mysql_connect($host, $user, $password )
              or die("Hey loser, check your server connection.");

//make sure we're using the right database

mysql_select_db($database);
?>
```

Never (never, never never) connect to the database as a superuser or as the database owner. Use always customized users with very limited privileges.

Do not print out any database specific information, especially about the schema, by fair means or foul. Which also means switch off error messages - even these can give clues to an attacker.

(See http://www.unixwiz.net/techtips/sql-injection.html)

Check if the given input has the expected data type. Always try to validate input before using it.

If the database doesn't support binding variables then quote each non-numeric user supplied value that is passed to the database with the database-specific string escape function such as mysql_real_escape_string(). Generic functions like addslashes() are useful only in a very specific environment.

Use prepared statements with bound variables. They are provided by PDO, by MySQLi and by other libraries.

The MySQL_ library is now officially deprecated and users are advised on the PHP website to use MySQLi(mproved). It is object-oriented but it provides a procedural interface for compatibility with MySQL_ .

MagicQuotes and Escape Characters

As of PHP v3.06, PHP had a configuration setting called 'magic_quotes'. If on, PHP would automatically add slashes to any escapable character coming in via HTTP get or post requests, and via cookies.

You could also do this manually in your code using the addslashes() function, if magic_quotes was switched off.

So you would end up with something like:

```
SELECT * FROM person
WHERE name = 'Reg' AND password = 'x\' or \'y\'=\'y'
```

This escaped any quotation marks, which meant that PHP would always treat them as part of the data string, and not part of the SQL query.

Unfortunately, due to differing server configurations, it's messy, inconsistent and unreliable.

mysqli_real_escape_string( )

An improvement to the problem of consistent treatment of escape characters.

If you are using the mysqli library, then you can use a function called:

```
mysqli_real_escape_string( );
```

This is tailored to the character set of the database connection and takes care of escaping the necessary characters before they are submitted to the database.

```
$connect = new mysqli($host, $user, $password, $database );

$string = "O'Reilly";

$safestring = mysqli_real_escape_string( $connect, $string );

$sql = "SELECT * FROM person WHERE name =' ".$safestring." ' ";
```

Prepared Statements

aka Parameterised statements. Also see 'bound variables'
(See http://www.php.net/manual/en/mysqli.quickstart.prepared-statements.php

```
$variable = "O'Reilly";

// prepare the query
$query = $mysqli->prepare( "SELECT x, y, z FROM tablename WHERE user = ?" );

// bind a parameter - here the first parameter is a short string that specifies the type that the
// subsequent arguments should be:
// 's' means a string    'd' means a double     'i' means an integer     'b' is a blob

$query->bind_param( 's',  $variable );

// execute query:
$query->execute( );

// so if we had a more complex query, which updated the user info with
// "favorite_color" (a string), "age" ( an integer ) and "description", a blob:

$query = $mysqli->prepare( "UPDATE tablename
                           SET favorite_color = ?, age = ?, description = ?
                           WHERE user = ?" );

// we would have a bind looking like this:
$query->bind_param( 'sibs', 'red', 27, $some_blob, $variable );
$query->execute();
```

Basic Security

Database

- Do not allow the website to connect to the database as the root / admin user.
- Set up a database user account, called something like 'webuser' and give it a password.
- Give that account the minimum privileges required to do the job.
- The website should only connect to the database with that username and password.

Website

- Do not use out of date libraries or commands.
- The original PHP MySQL library has been deprecated for a reason.
- Never trust any kind of input, especially that which comes from the client side, even though it comes from a select box, a hidden input field or a cookie.
- Your PHP code should always validate the input wherever possible.