Name: _____ **Student ID:**_____

You will need:
RedHat Linux 7.3 (VM)
Windows XP (VM)
A computer with internet access

COS80013 Internet Security

Lab 9

In this lab you will experiment with SQL injection.

1. Using Virtual Machine Launcher, start up the ***RedHat Linux with local network*** VM image.
Also start up the ***Windows XP with local network*** image.

2. Open a browser on the XP image. Surf to
http://www.server.com/safelogin.php (or http://192.168.100.104/safelogin.php)

Log in as
`Warren` *(user)*
`Eclipse` *(pass)*

and have a look around the application. Don't change anything.

Log Out.

The simplest form of SQL injection involves passing unauthorised SQL instructions to the SQL engine on the database management system server, i.e. sending executable SQL script to the database through a data field on a web page.
The Swinburne Usability Lab booking system on the RHLinux VM image has been altered to make it partially susceptible to SQL injection attacks. The PHP module has been altered to allow the ' character to be passed to the database engine ('magic quotes' has been turned off). The original ***login.php*** page has been altered to allow input fields of any size.

The original ***safelogin.php*** page encrypts the password field before it sends it to the database. The weakened version (***login.php***) does not.

3. In the browser, go to
http://www.server.com/login.php
(or http://192.168.100.104/login.php)

Try logging in as ***Warren*** again.

The password won't work because the string 'Eclipse' has not been hashed, but the password in Warren's part of the database is in a hashed form (something like '10550f076f688b75').

Try logging in as Warren again, but use the following string as the password:

`x' or 'x' = 'x` (not Warren's password)

You are now logged it!

Name: _____ **Student ID:**_____

Why does it work?

If we could look at the source of the ***login.php*** file we would find some code that looks like this:

```
SQLString = "SELECT Username FROM Users WHERE Username =
'$User' AND Password = '$Pass';"
```

When the page is submitted, the values for '$User' and '$Pass' are replaced by the text you type into the username and password fields on the page. When you type

```
Warren
 x' or 'x' = 'x
```

what gets sent to the SQL engine is:

```
SELECT Username FROM Users
WHERE Username = 'Warren'
AND Password = 'x' or 'x' = 'x';
```

(true if Warren is in the database)

false OR true = true

4. Log out and return to
http://www.server.com/login.php

Another way of getting a WHERE statement to return **true** is the use this string:

```
' or 1=1;--
```

This won't work with login.php because 'smart hyphens' have not been disabled.

Try it. You can use

```
x' or 'x' = 'x
```

for the user (true for any user) and

```
' or 1=1;--
```

for the password. **It won't work**, but on some web sites, it will!

5. Login pages are normally locked-down so much that the above scripts won't work. However, most web sites have a password reminder form that will accept an e-mail address and send the password to you by (plain text) e-mail. These forms are less filtered because e-mail addresses can be **any length** and can contain a broad range of characters including @ ' – and ; .
An attacker can use these forms to guess (by trial and error) the names of columns and tables in the database, as well as running multiple ("stacked") queries, sub-queries and UNIONed queries. This is called **Blind Injection**

Let's do some **schema mapping**.
The key to schema mapping is distinguishing between DBMS error messages and web server (or web page) error messages. If you don't get an error from the DBMS that means that the SQL you injected worked correctly – *i.e.* the column name or table name was found in the database.

Name: _____ Student ID:_____

From the **login.php** page, enter in both fields:
`x' and user = 'x`

How many errors?

If the user's name column is not called 'user', you will get an error from the **database server** and you will get a polite message from the web application.
If the column name matches, you will only get the message from the web application.
Try it again with
x' and username = 'x

How many errors?

username is a valid column name in the database table!

Table names can be guessed in a similar way – just add the table name to a found column name using the dot syntax:

x' and users.username = 'x
Try
`x' and ACCOUNT.username = 'x`

A good place to find guesses for column names is to look at pages (especially html pages) with form fields on them.
Use your OO or database modelling skills to guess the table names.
Some frameworks like *Ruby on Rails* map the columns to the web text field names automatically.

Guessing data

6. Guessing user names can be done using keywords such as LIKE.

`x' or username LIKE '%admin%`

What happens? What does it mean?

7. If a user has forgotten to set their password, you can try:

Name: _____ **Student ID:**_____

```
x' or password = '
x' or password is NULL--'
```

true if the user has not entered a password yet
works in some systems.

8. If we were using a form / scripting language / DBMS that was even more mis-configuration, we would be able to execute queries like these:

Note: these won't work on this server – it's not that badly set up.

```
x' and 1=(select count(*) from tablename); --
```
guess table name without knowing column name

```
x'; insert into ACCOUNT (username) values ('hacker');
```
add user

```
x'; update ACCOUNT set password = 'hacker' where username
= 'admin
```
change password

```
x' ; drop table ACCOUNT; --
```
create havoc

```
function sanitize($input)  //you can nest these php sanitizing functions thus:
{
    return nl2br(strip_tags(stripslashes(htmlentities(htmlspecialchars($input)))));
}
```

9.
On the host PC, go to
http://www.unixwiz.net/techtips/sql-injection.html

and read the section on 'Mitigations'

What are the seven things you can do to prevent SQL injection?

1.

2.

3.

4.

5.

6.

7.

Shutdown all the VMWare machines and log out.

End of Lab