**Part1 Security**

**Section i**

An SQL injection attack is where an attacker uses flaws in the security of an application which generates an SQL statement to be processed by a DBMS, to insert their own SQL queries, thus viewing data they shouldn’t be able to, inserting their own data or modifying / deleting the existing data. The extent to what the attacker can do will be limited by the database security protocols like access rights, but in a non-secure database, these attacks could have disastrous consequences. It is basically achieved by using a character with meaning in SQL, like a single quote, where a simple string of alphanumeric characters is intended, to allow the rest of the input to add SQL language to the query.

In the case of this example code, whatever the user inputs for the “username” variable is included between single quotes as the value in the where clause of the generated statement. On a very basic level, the user could enter any username and get access to that record since usernames are often easily discoverable and/or guessable.

On a more severe level, as there are no restrictions on what text could put into this username value, entire SQL queries could be added onto the end.

For example a user input of …

“mary01’; SELECT \* FROM MOD\_ENROL; --”

would allow the user to see all of the data in the MOD\_ENROL table, not just that of one user. Here the input takes advantage of some SQL databases like Microsoft’s SQL Server ability to execute multiple queries separated by ; characters. It completes original query with a username, and then issues another query to see all the data in that table. The comments character at the end will comment-out the trailing single quote, preventing a syntax error in the injected code.

The user could use this technique to enter any number of queries doing a variety of tasks. Accessing data could lead to identity theft, but inserting, modifying or deleting data could have other serious repercussions too. In theory using this technique, the user could have access to the entire database, depending on what security is in place on the database side.

**Section ii**

In the case mentioned, the Injection attack would still be possible, even for someone with a basic knowledge of SQL. Not only is the ability to insert code via the variable is not addressed, but in fact the password requirement in the where clause could be easily circumvented by using block comments, the opening one at the end of the username input, and the closing one in the password field. So if the newly generated querystring would be something like:

SELECT YEAR, MOD\_CODE FROM MOD\_ENROL WHERE STUDENT = ‘username’ AND PASSWORD = ‘password’.

The username variable was “mary01’; <<SQL QUERIES HERE>>; /\*” and the password variable was “\*/ --“

The generated SQL query would read:

SELECT YEAR, MOD\_CODE

FROM MOD\_ENROL

WHERE STUDENT = ‘mary01’;

<<NEW QUERIES HERE>>;

/\*AND PASSWORD = ‘\*/ --‘

Here the same injection attack has occurred and in fact the ability to see mary01’s details is still possible even without the password.

**Section iii**