1. SQL Injection

Vulnerability/Risk	
High SQL Injection occurs when the attacker provides untrusted input into a vulnerable application, which gets processed by an interpreter. The input is directly included in an SQL query, thus allowing the attacker to alter the course of execution. SQL Injection flaws are very common and widespread. Attackers are able to perform such an attack because the user's input is not validated, filtered or sanitized. Since attackers are able to alter the queries to be executed on a database server, SQL Injection can expose entries that are inside the database and allow them to have unauthorised access to pages without knowing the actual username and password. SQL Injection can also cause loss of data, corruption, and even denial of access. SQL Injection can also at times, lead to complete host takeover by the attackers. This web application is vulnerable to SQL injection whereby attackers can find out user's credentials and gain access to the administrator's account.	

Detailed Example

commonDB.java

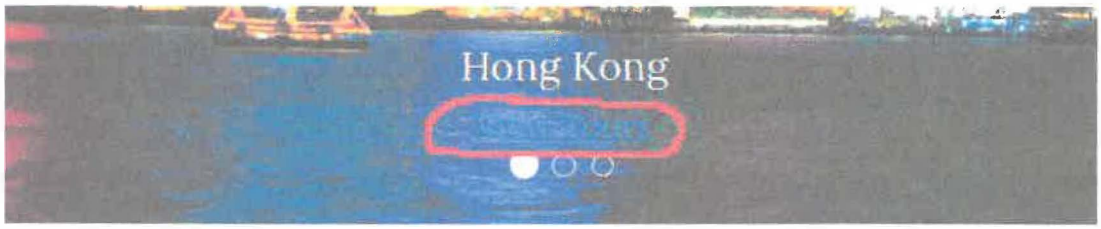
The vulnerability in this file allows attackers to gather information on every single user in the database. In this file, prepared statements were not used when writing a query to the database. The code below exposes the SQL injection flaw whereby prepared statements were not used. Malicious user input will be treated as part of the query and executed.

```
String searchedDetails = "";

String country = countrySelected;
Class.forName("com.mysql.jdbc.Driver");
String connURL = "jdbc:mysql://localhost/sptravel?user=root&password=password";
Connection conn = DriverManager.getConnection(connURL);
Statement stmt = conn.createStatement();
String sql = "SELECT * FROM sptravel.tour_pkg ";
sql += "WHERE Country='" + country + "' AND Travel_Month like '"
        + dateSelected + "%' AND Min_Cost <= '" + maxCost + "'";

System.out.println(country+" "+dateSelected+" "+maxCost);
ResultSet rs = stmt.executeQuery(sql);
```

Due to this, it is possible to launch an SQL Injection attack through the **index.jsp** page.



When the attacker clicks on "View Tours", he will be redirected to **TourList.jsp** where the search results are displayed. However, since the web parameters were passed using the GET function, it is displayed along with the URL as shown.

```
<div class="carousel-caption">
    Hong Kong <br> <a
        href="common/TourList.jsp?Country=HK&Date=&Price=999999"><small>View
            Tours</small></a>
```

As the parameters are displayed to him in clear sight, he can easily direct input into the database queries. After a couple of queries using the Information_Schema table to find out more information about the database, he can launch an SQL Injection attack to find out details of every user in the database, including the administrator.

```
urList.jsp?Country=HK' or 1=1 union select Account_ID,Username,1,Acc_Level,Password,Email from accounts;-- -
```

The above query displays the following results,

Now the attacker knows the Administrator's username and password.

Recommendations

The main cause of SQL injection attacks is due to the lack of use of prepared statements. This is because when a SQL server receives query, it goes through a couple of stages first. The first few stages are for compiling the query, where keywords used in the query are converted to machine readable format. After the compilation stage, it then enters the execution stage where the query is executed. If an application is using regular statements, the entire query would be executed just like that, but with prepared statements and parameterized data (the "?" character in prepared statements), data will only be supplied right before the execution stage. Data supplied here will be treated as pure data only, meaning that it cannot be executed. Therefore, we highly recommend that in order to prevent SQL Injection, the programmer should use parameterized queries (JDBC Prepared statements) and use frameworks replacing parameters via setXXX in the **commonDB.java** page. The following code shows how the prepared statements will be used.

```
String sql = "SELECT * FROM sptravel.tour_pkg WHERE Country= ? AND "
        + "Travel_Month like ? AND Min_Cost <= ?";
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setString(1, countrySelected);
pstmt.setString(2, "%"+dateSelected+"%");
pstmt.setString(3, maxCost);
System.out.println(country+" "+dateSelected+" "+maxCost);
ResultSet rs = pstmt.executeQuery();
```