CS 542 – Introduction to Software Security
Exercise on SQL Injection

Binhao Chen (bchen276@wisc.edu), Steven Yang (yang558@wisc.edu)

Due: October 13 at 2:30pm

1 For the SQL injection in Java

1.1 Screenshots showing the input used for the attack, and the output you got from the system

```
user@software-security22:~/Desktop/EXERCISES/3.8.1_sql_injections$ make
Compiling exercise program...
user@software-security22:~/Desktop/EXERCISES/3.8.1_sql_injections$ java Main
username: some_guy
password: his_password
Login Successful! Welcome some_guy

username: some_guy
password: 123456
Login Failure.

username: some_guy
password: ' OR 'a'='a
Login Successful! Welcome some_guy

username: binhao
password: ' OR 'a'='a
Login Successful! Welcome binhao

username:
```

1.2 Your commented code for the mitigation

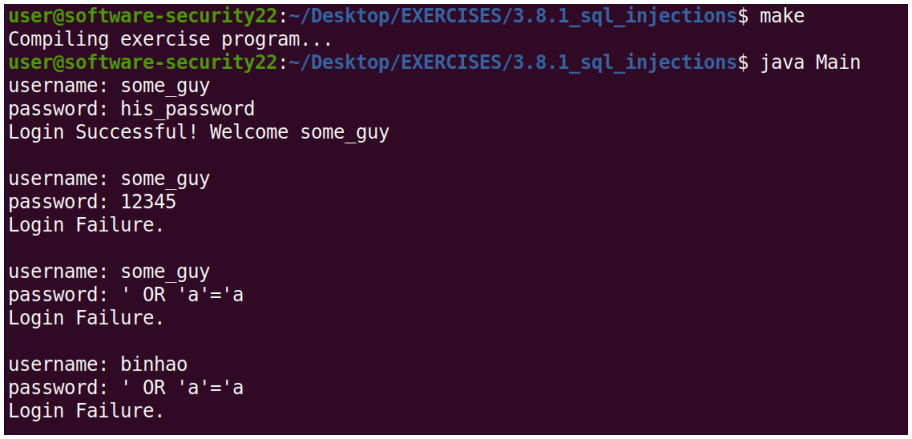
```
1
2  import java.io.Console;
3  import java.sql.Connection;
4  import java.sql.DriverManager;
5  import java.sql.ResultSet;
6  import java.sql.SQLException;
7  import java.sql.Statement;
8  import java.sql.PreparedStatement;
9
10 /**
11  * Main execution class for sql_injection exercise. Prompts user for username
12  * and password to lookup in the accompanying sqlite3 database.
13  *
14  * @author Joseph Eichenhofer
15  *
16  */
17 public class Main {
18
19     private static final String DB_URL = "jdbc:sqlite:users.db";
20
```

```
21      /**
22       * Prompt user for username and password. Displays login success or failure
23       * based on lookup in user database.
24       *
25       * @param args
26       *            n/a
27       */
28      public static void main(String[] args) {
29          Console terminal = System.console();
30
31          if (terminal == null) {
32              System.out.println("Error fetching console. Are you running from an
                     IDE?");
33              System.exit(-1);
34          }
35
36          while (true) {
37              // get username and password from user
38              String username = terminal.readLine("username: ");
39              if (username.toLowerCase().equals("exit"))
40                  break;
41              String password = terminal.readLine("password: ");
42
43              // check username and password
44              boolean loginSuccess = false;
45              try {
46                  loginSuccess = checkPW(username, password);
47              } catch (SQLException e) {
48                  System.out.println("Database Error.");
49                  e.printStackTrace();
50              }
51
52              if (loginSuccess)
53                  System.out.println("Login Successful! Welcome " + username);
54              else
55                  System.out.println("Login Failure.");
56
57              // separate iterations for repeated attempts
58              System.out.println();
59          }
60      }
61
62      /**
63       * Connect to the sample database and check the supplied username and
             password.
64       *
65       * @param username
66       *            username to check
67       * @param password
68       *            password to check for given username
69       * @return true iff the database has an entry matching username and
             password
70       * @throws SQLException
71       *             if unable to access the database
72       */
73      private static boolean checkPW(String username, String password) throws
             SQLException {
74          // declare database resources
75          Connection c = null;
76          Statement statement = null;
77          ResultSet results = null;
78
79          try {
```

```
80              // connect to the database
81              c = DriverManager.getConnection(DB_URL);
82
83              // check for the username/password in database
84              // String sqlQuery = "SELECT COUNT(*) AS count FROM USERS WHERE
                    username == '" + username
85              //      + "' AND password == '" + password + "'";
86              // statement = c.createStatement();
87              // results = statement.executeQuery(sqlQuery);
88
89              // This code block is how we did to mitigate the SQL Injection
                    Vulunerability.
90              // We use the PreparedStatement module by importing java.sql.
                    PreparedStatement;
91
92              // This will reserve spaces in the query statement for data input
                    and make the SQL parse the original query without the input
                    data from the user. And then later compare the input from the
                    user with the database we have.
93
94              // We put two question mark placeholders for username and password
                    in the query. Then, we set the input values with setString().
95
96              //In this way, the input is not parsed so that the user is not
                    allowed to interact with the SQL query. The malicious input is
                    simply a strange string.
97
98              PreparedStatement pstmt = c.prepareStatement("SELECT COUNT(*) AS
                    count FROM USERS WHERE username = ? AND password = ?");
99              pstmt.setString(1, username);
100             pstmt.setString(2, password);
101             results = pstmt.executeQuery();
102
103             // if no user with that username/password, return false;
104             //otherwise must be true
105             //if (results.getInt("count") == 0)
106             //   return false;
107             //else
108             //   return true;
109
110             // This is part of our mitigation solution:
111             // Here we rewrite this if clause to ensure the return value from
                    SQL Query is 1,
112             //to prevent any other cases like the corruption of database.
113             //In this way, only when there exists exactly one row that
                    corresponds to the
114             //input username+password can login the system.
115             if (results.getInt("count") == 1)
116                 return true;
117             else
118                 return false;
119
120
121         } finally {
122             // release database resources (ignore any exceptions including null
                     pointer)
123             try {
124                 results.close();
125             } catch (Exception e) {
126             }
127             try {
128                 statement.close();
129             } catch (Exception e) {
```

```
130                 }
131                 try {
132                     c.close();
133                 } catch (Exception e) {
134                 }
135             }
136         }
137 }
```

1.3 Screenshots showing the attack input(s) and fixed output after fixing the vulnerability, for both "good" and malicious input



1.4 An explanation on your attack and your mitigation

Attack: To be able to attack the system, we can enter a password that, when inserted into the SQL query, will ensure that the WHERE clause is always satisfied. We can achieve it by passing in **' OR 'a' = 'a**. The first single quote will match the one in the query and allow us to introduce a new OR logic term. The statement 'a' = 'a' will always be true. Therefore, the WHERE clause is always true, the returned count will never be 0 and the SQL query will always be executed without errors. Then the checkPW function will return true.

Mitigation: We mitigate by using the Prepared Statement. We put two question mark placeholders for username and password in the query. Then, we set the input values with setString() function. In this way, the input is not parsed so that the user is not allowed to interact with the SQL query. The malicious input will only be treated as a strange string. Also, we rewrite the code that check "if (results.getInt("count") == 1)" to ensure the return value from SQL Query is 1, to prevent any other cases like the corruption of database. In this way, only when there exists exactly one row that corresponds to the input username+password can login the system.

2 For the SQL injection in Python

2.1 Screenshots showing the input used for the attack in the vulnerable version, and the output you got from the system

```
user@software-security22:~/Desktop/EXERCISES/3.8.1_sql_injections$ cd inPython
user@software-security22:~/Desktop/EXERCISES/3.8.1_sql_injections/inPython$ ls
create.py  mydb  sqlMain.py
user@software-security22:~/Desktop/EXERCISES/3.8.1_sql_injections/inPython$ nano sqlMain.py
user@software-security22:~/Desktop/EXERCISES/3.8.1_sql_injections/inPython$ python sqlMain.py

 username: some_guy

 password: his_password
2.6.0
Login Successful! Welcome  some_guy

 username: some_guy

 password: 123
2.6.0
Login Failure.

 username: some_guy

 password: ' OR 'a'='a
2.6.0
Login Successful! Welcome  some_guy

 username: binhao

 password: 'OR 'a'='a
2.6.0
Login Successful! Welcome  binhao
```

2.2 Your commented code with and without prepared statements

Below is the revised copy of the **sqlMain.py** file.
The attack code and comments are highlighted by the orange pen:

```
1  import sqlite3
2  from sqlite3 import Error
3
4  def create_connection(db_file):
5      """ create a database connection to a SQLite database """
6      conn = None
7      try:
8          conn = sqlite3.connect(db_file)
9          print(sqlite3.version)
10         return conn
11     except Error as e:
12         print(e)
13
14     return conn
15
16
17
18 def checkPW(u, p):
19
20 #This is the method you need to implement.
21 #It has to have the SAME functionality as the exercise in Java.
22 #The first version has to be vulnerable to SQL injection attacks,
23 #and the second version must use prepared statements to mitigate
24 #that attack.
25 #
26 #The parameters are:
27 #u: username
28 #p: password
29 #
30 #Return value:
31 #True if the login attempt was successful.
32 #False otherwise.
33
```

```
    conn = create_connection(r"/home/user/Desktop/EXERCISES/3.8.1_sql_injections/inPython/mydb/pythonsqlite.db")
    cur = conn.cursor()

    # This string concatenation is used for implementing the checkPW
    # with the same functionality as the SQL injection exercise
    # in Java. This version will not use prepared statements
    # and will be vulnerable to attack.
    #sqlQuery = f"SELECT COUNT(*) AS count FROM USERS WHERE login == '{u}' AND password == '{p}' "

    # This code block is how we did to mitigate the SQL Injection Vulunerability.
    # This prepared statement (bu using the question mark) will reserve spaces
    # in the query statement for data input and make the SQL parse the original query without the input data from the user.
    # And then later compare the input from the user with the database we have.

    # We use question mark placeholders for login and password in the query.
    # In Python, we can pass in a tuple of login and password to the
    # cursor.execute() method to set the input values.
    # Then we use fetchall() to get the output (an array of tuples)
    # and use indexing to get the count.

    sqlQuery = '''SELECT COUNT(*) AS count FROM USERS WHERE login == ? AND password == ?'''
    input = (u, p)

    try:
        #cur.execute(sqlQuery)
        cur.execute(sqlQuery, input)
        results = cur.fetchall()
        num_of_row = results[0][0]

        #if (num_of_row == 0):
        #    return False
        #else:
        #    return True

        # This is part of our mitigation solution:
        # Here we rewrite this if clause to ensure the return value from SQL Query is 1,
    #to prevent any other cases like the corruption of database.
    #In this way, only when there exists exactly one row that corresponds to the
    #input username+password can login the system.
        if (num_of_row == 1):
            return True
        else:
            return False

    finally:
        try:
            conn.close()
        except:
            print("Something went wrong")


if __name__ == '__main__':

    while 1:
       username = input("\n username: ")
       if username == "exit":
          quit()
       password = input("\n password: ")

       loginSuccess = False

       try:
          loginSuccess = checkPW(username, password)
       except:
          print("Something went wrong")

       if (loginSuccess):
           print("Login Successful! Welcome ",  username)
       else:
```

```
103             print("Login Failure.")
```

2.3 Screenshots showing the attack input(s) and fixed output after fixing the vulnerability, for both "good" and malicious input

```
user@software-security22:~/Desktop/EXERCISES/3.8.1_sql_injections/inPython$ nano sqlMain.py
user@software-security22:~/Desktop/EXERCISES/3.8.1_sql_injections/inPython$ python sqlMain.py

 username: some_guy

 password: his_password
2.6.0
Login Successful! Welcome  some_guy

 username: some_guy

 password: 123456
2.6.0
Login Failure.

 username: some_guy

 password: 'OR 'a'='a
2.6.0
Login Failure.

 username: binhao

 password: 'OR 'a'='a
2.6.0
Login Failure.
```

2.4 An explanation on your attack and your mitigation

Attack: To be able to attack the system, we can enter a password that, when inserted into the SQL query, will ensure that the WHERE clause is always satisfied. We can achieve it by passing in **' OR 'a' = 'a**. The first single quote will match the one in the query and allow us to introduce a new OR logic term. The statement 'a' = 'a' will always be true. Therefore, the WHERE clause is always true, the returned count will never be 0 and the SQL query will always be executed without errors. Then the checkPW function will return true.

Mitigation: Similarly, we use question mark placeholders for login and password in the query. In Python, we can pass in a tuple of login and password to the cursor.execute() method to set the input values. Then we use fetchall() to get the output (an array of tuples) and use indexing to get the count. In this way, the attacker will not be able to interact with the query so that the malicious input will only be treated as a strange string. Also, we rewrite the code "if (num of row == 1)" to ensure the return value from SQL Query is 1, to prevent any other cases like the corruption of database. In this way, only when there exists exactly one row that corresponds to the input username+password can login the system.