NOTE: SCENARIO IS WHAT PUZZLE TAKER SEES.

SCENARIO:

You are developing a web application requiring authentication and authorization. The persistent store used for credential information (usernames and passwords) is a SQL database. The database has a table called “users” with three columns: “username” (the primary key), “password”, and “roles”. You are asked to write a helper method to get a username (a String value) and return its record containing roles information to the caller. Consider the snippet of code below and answer the following questions, assuming that the code has all required permissions to execute.

```
// OMITTED: Import whatever is needed.
public final class DbUtils {
  public static ResultSet getUser (String username)
      throws SQLException {
    String sql = "SELECT * FROM users WHERE username = '"
        + username.toLowerCase() + "';";

    // Get a connection from the connection pool:
    Connection conn = DbUtils.getConnectionPool().getConnection();
    PreparedStatement stmt = conn.prepareStatement(sql);
    return stmt.executeQuery();
  }
}
```

Questions:
1. What will the getUser method do when executed?

2. If one calls the getUser method with an arbitrary String value given as username, which one statement is correct?

a. The method reads and returns the roles field.
b. The method may read and return the username, roles, and password fields.
c. The method may read and return information from all users.
d. The method may read and write information from all users.
e. None of the above.

[Other statistical questions will be imported here while creating survey.]

NOTE: ANSWER IS TO BE SHOWN TO PUZZLE TAKER AT THE END OF SESSION.

ANSWER:

b
Some programmers wrongly think that by using the `PreparedStatement` class, the SQL injection vulnerability can be resolved. The `PreparedStatement` class can resolve SQL injection vulnerability only if it is used correctly. Using string concatenation to build SQL query is highly vulnerable to SQL injection. In this implementation the query is still built by concatenation.

NOTE: THE REST OF DOCUMENT CONTAINS EXTRA INFORMATION FOR THE PROJECT RESEARCHERS. IT IS NOT TO BE SHOWN TO PUZZLE TAKERS.

TAGS:
java, preparedstatement-class, sql-injection, input-verification, input-sanitization

CATEGORIES:
Blindspot - YES
Type - SQL
Number of distinct functions - 5
Number of total functions - 5
Blindspot function - `PreparedStatement.setString()`
Function call omitted - YES
Blindspot type - Omitted function call
Number of Parameters in the blindspot function - 2 parameters
Cyclomatic complexity - 2

NAME:
PreparedStatement class - An object that represents a precompiled SQL statement.

DESCRIPTION:
The simplest way to perform SQL queries in Java is to build a string as a SQL statement and then pass it to a `Statement` object to be executed. The other way is to build the query by using `PreparedStatement` which compile the query before the execution.

BLINDSPOT:
The fact that the `PreparedStatement` class can resolve SQL injection vulnerability only if it is used correctly, may cause a blindspot. The simplest way to build SQL queries is string concatenation. This simple way of performing SQL queries is highly vulnerable to SQL injection attack even if the `PreparedStatement` class is being used to perform the query.

CORRECT USE EXAMPLE:

```java
import java.sql.*;
public class Authenticator {
  public static boolean authenticate ( String username, char[] passwd )
      throws SQLException {
```

```
    String sql = "SELECT * FROM users WHERE " +
        "username = ? AND password = ?;";
    Connection conn = getConnectionPool().getConnection();
    PreparedStatement stmt = con.prepareStatement(sql);
    stmt.setString(1, username);
    stmt.setString(2, calcHash(passwd));
    ResultSet rs = stmt.executeQuery();
    // Close statement is omitted!
    return rs.next();
  }
}
```

MORE INFORMATION:
To see the correct way of use of the API look at the JX57-sql-injection puzzle.

REFERENCES:
1. https://www.owasp.org/index.php/SQL_Injection