CSCE 5565 Quiz 5 - Fall 2023

Student name:

Question 1 (10 points)

Web developers often use client-side controls in their web applications, similar to the one used in the code below.

```
<form method="post" action="Shop.aspx?prod=2" onsubmit="return validateForm(this)">
Product: Samsung Multiverse <br/>
Price: 399 <br/>
Quantity: <input type="text" name="quantity"> (Maximum quantity is 50) <br/>
<input type="submit" value="Buy"> </form>
<script>
function validateForm(theForm) {
    var isInteger = /^\d+/;
    var valid = isInteger.test(quantity) && quantity > 0 && quantity <= 50;
    if (!valid)
        alert('Please enter a valid quantity');
    return valid;
}
</script>
```

Questions:

1. Specify the steps to bypass the control. (5 points)

To bypass the client-side validation control in the provided code, you would need to either manipulate the client-side code execution or directly make a request to the server without using the provided form.

Disable JavaScript
Edit HTML
Manipulate Form Data

2. Propose an alternative solution to enforce sensitive controls. (5 points)

The control need to on the server side.

Exercise 1 (10 points)

Consider the following Java code snippet responsible for key generation and encryption using the Java Cryptography Architecture (JCA):

```
 1 // Key generation code
 2 SecureRandom random = new SecureRandom();
 3 byte[] salt = new byte[32];
 4 random.nextBytes(salt);
 5 PBEKeySpec spec = new PBEKeySpec(pwdChar, salt, 500, 128);
 6 SecretKeyFactory skf = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
 7 SecretKey key = skf.generateSecret(spec);
 8 
 9 // Encryption code
10 Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
11 cipher.init(Cipher.ENCRYPT_MODE, key);
12 byte[] cipherText = cipher.doFinal(inputMsg);
```

Questions:

1. Identify the cryptographic vulnerability in the code and provide the line number. (5 points)

Answer: Line 5

2. Propose a solution to fix the vulnerability. (5 points)

   Answer: change value from 500 to 1000 or above.

Exercise 2 (20 points)

Consider the following Java code snippet:

```
1  protected void processRequest(HttpServletRequest request,
2                            HttpServletResponse  response)
3                             throws ServletException, IOException {
4          try {
5              String EmployeeId = request.getParameter("EmpId");
6              Connection conn = null;
7              try {
8                  Class.forName(driver).newInstance();
9                  conn  =  DriverManager.getConnection(url  +  dbName,
10                                           userName, password);
11                 Statement st = conn.createStatement();
12                 String query = "SELECT * FROM  Staff where
13                                          EmployeeId='" + EmployeeId + "'";
14                 ResultSet res = st.executeQuery(query);
15                 PrintWriter out = response.getWriter();
16                 while (res.next()) {
17                     String s = res.getString('Salary');
18                     out.println('\t\t' + s);
19                 }
20                 conn.close();
21             } catch (Exception e) {
22                 e.printStackTrace();
23             }
24         } finally {
25             out.close();
26         }
27     }
```

Questions:

1. Locate the code lines that introduce a SQL injection vulnerability. (5 points)

Answer: line 12 and 14 are correct.
[Line 5 and 9 are wrong]

2. Write a short script that utilizes a common code analysis method to identify the vulnerability. Specify the type of code analysis you used. (10 points)
   Ans: Taint analyis

3. Propose a solution to fix the vulnerability. (5 points)

Answer: Prepared statement.