A developer could easily repair this vulnerability by using a prepared return all the account numbers and balances it has stored.

And this results in accountBalanceQuery being:

\[
\text{SELECT accountNumber, balance FROM accounts WHERE account_owner_id = 0 OR 1=1}
\]

The attacker could change the parameter "user_id" to be interpreted as:

\[
\text{SELECT accountNumber, balance FROM accounts WHERE account_owner_id = 984}
\]

When you access your bank account online, the database query might look like this (in Java):

```java
try {
    Statement statement = connection.createStatement();
    String accountBalanceQuery = "SELECT accountNumber, balance FROM accounts WHERE account_owner_id = ?";
    PreparedStatement ps = connection.prepareStatement(accountBalanceQuery);
    ps.setInt(1, user_id);
    ResultSet rs = ps.executeQuery();
    while (rs.next()) {
        page.addTableRow(rs.getInt("accountNumber"), rs.getFloat("balance"));
    }
} catch (SQLException e) {
    // Handle exception
}
```

The result of SQL injection attacks. Here are a few recent examples and their consequences.

**SQL injection vulnerability, according to Veracode’s Security Rating for Applications (SRA):**

**WHAT IS SQL INJECTION?**

SQL injection is a high-severity vulnerability.

**ATTACKERS CAN EXPLOIT SQL INJECTION VULNERABILITIES TO:**

- Access or delete sensitive data
- Impersonate other users
- Control an application’s data-driven behavior
- Execute unexpected commands

**EXAMPLE: SQL INJECTION VULNERABILITY IN ACTION**

An attacker may have compromised an account and the following SQL statement is executed:

```sql
SELECT accountNumber, balance FROM accounts WHERE account_owner_id = 0 OR 1=1
```

The result might look like this:

```
<table>
<thead>
<tr>
<th>accountNumber</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>1000.00</td>
</tr>
<tr>
<td>67890</td>
<td>2000.00</td>
</tr>
</tbody>
</table>
```

**DEFEND AGAINST SQL INJECTION**

- Use prepared statements
- Sanitize user input
- Use parameterized queries
- Use ORM mappers
- Implement least privilege

**WHAT IT THE NEXT STEPS?**

- Learn more about SQL injection protection
- Automate the detection and prevention of SQL injection
- Take the Veracode Smart Developer Course to learn how to build secure software

**SIGN UP NOW!**