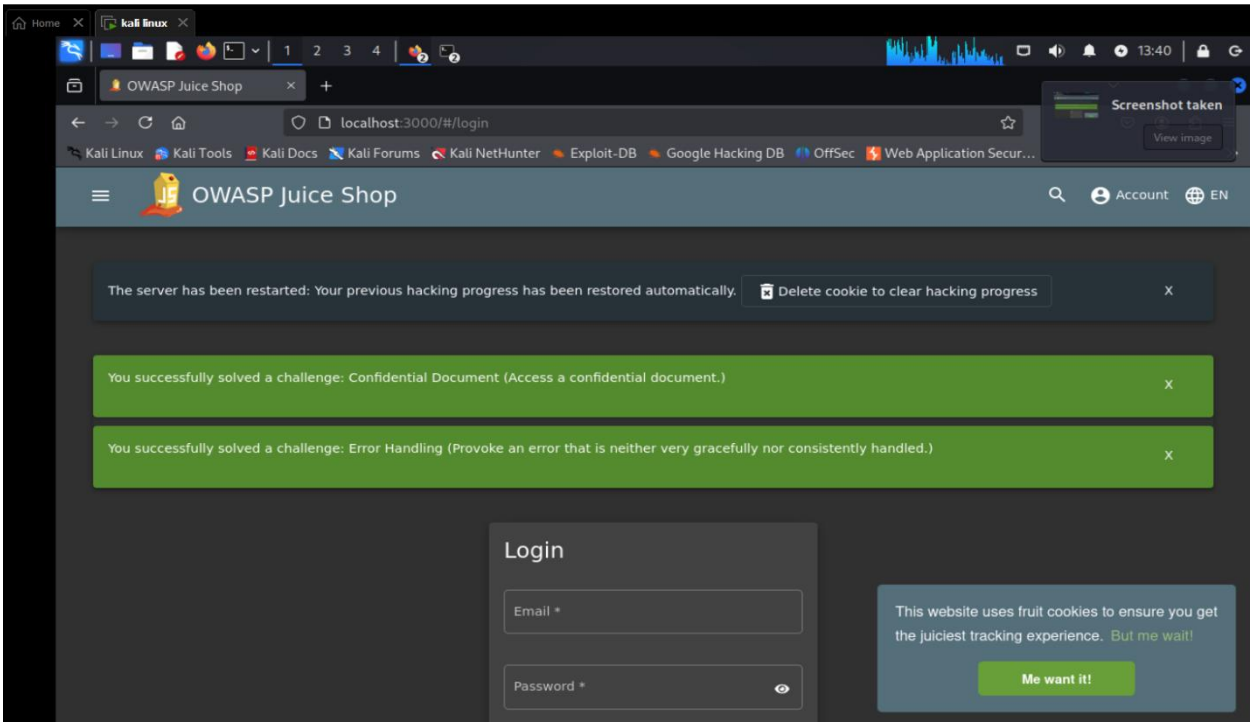
Security Testing Write-Up (OWASP Juice Shop)

1. SQL Injection via Login Bypass

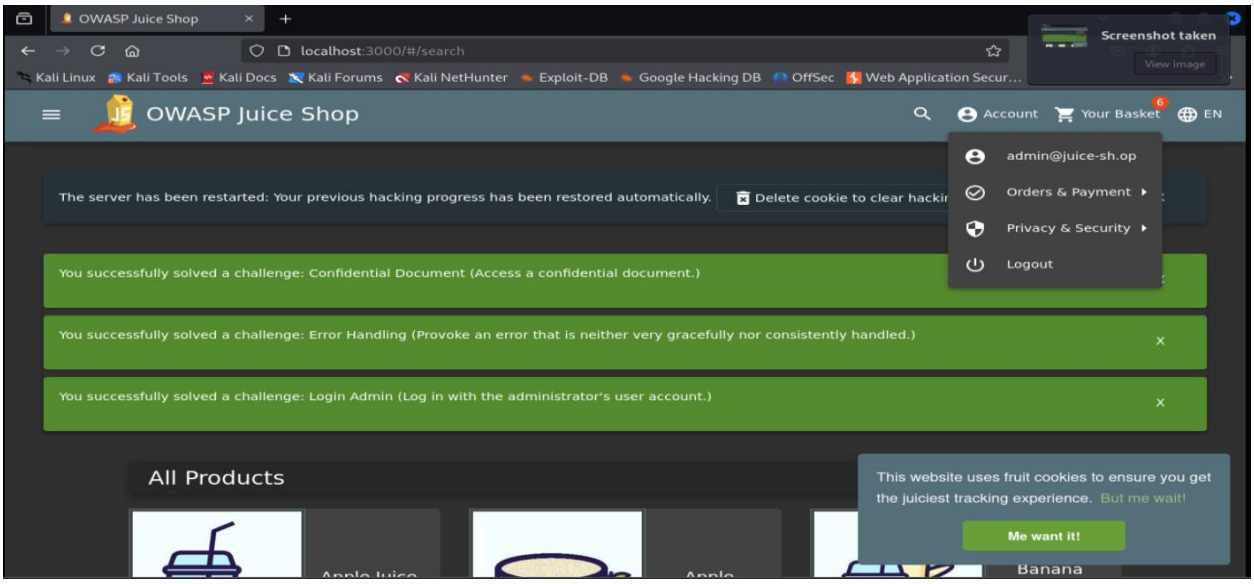
Goal: Gain unauthorized access by manipulating SQL queries in the login form.

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

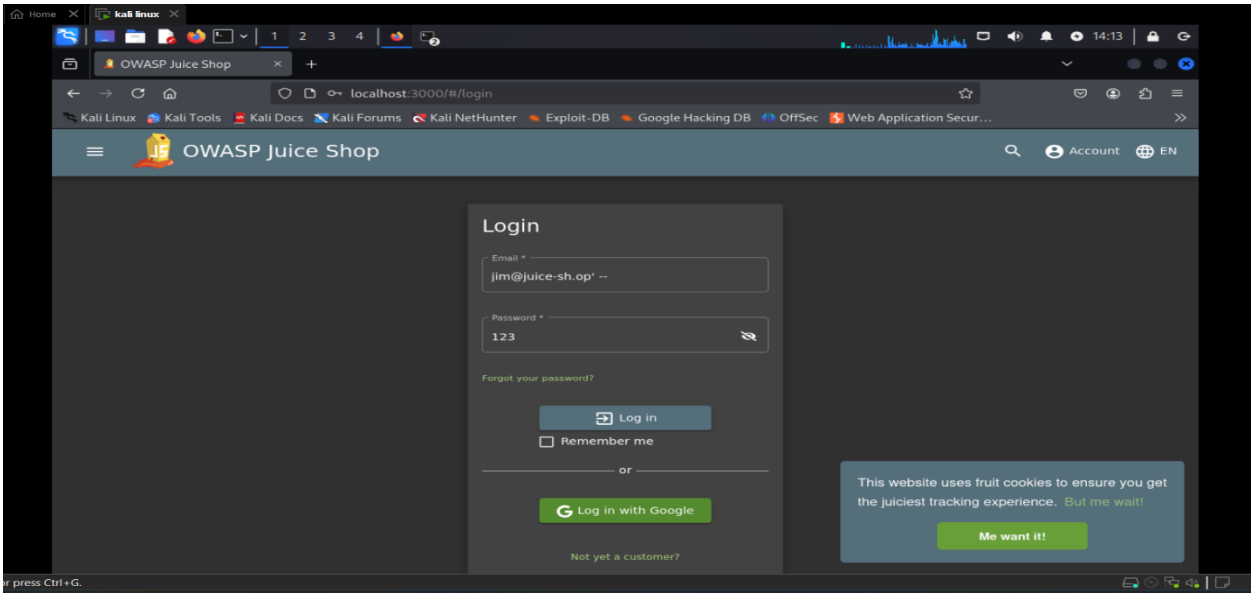
Log in with Jim's user account

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

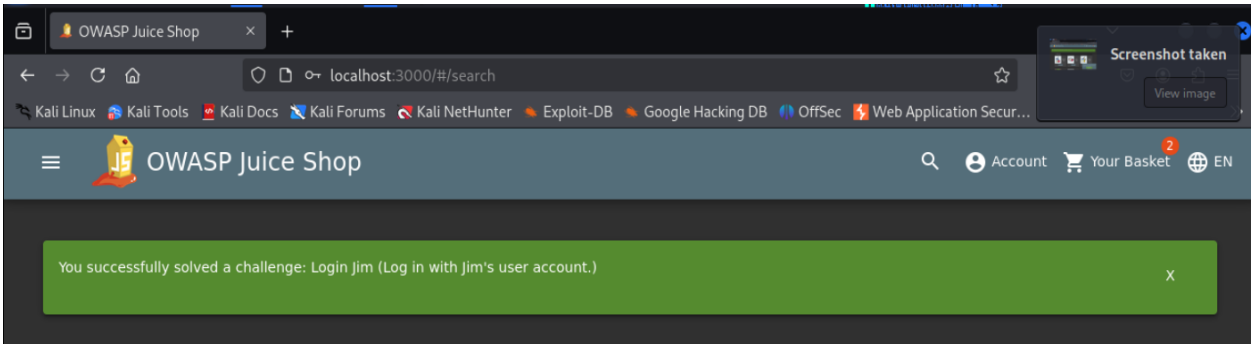
- The challenge description probably gave away what form you should attack.
- You need to know (or smart-guess) Jim's email address so you can launch a targeted attack.
- If you harvested Jim's password hash, you can try to attack that instead of using SQL Injection.

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.

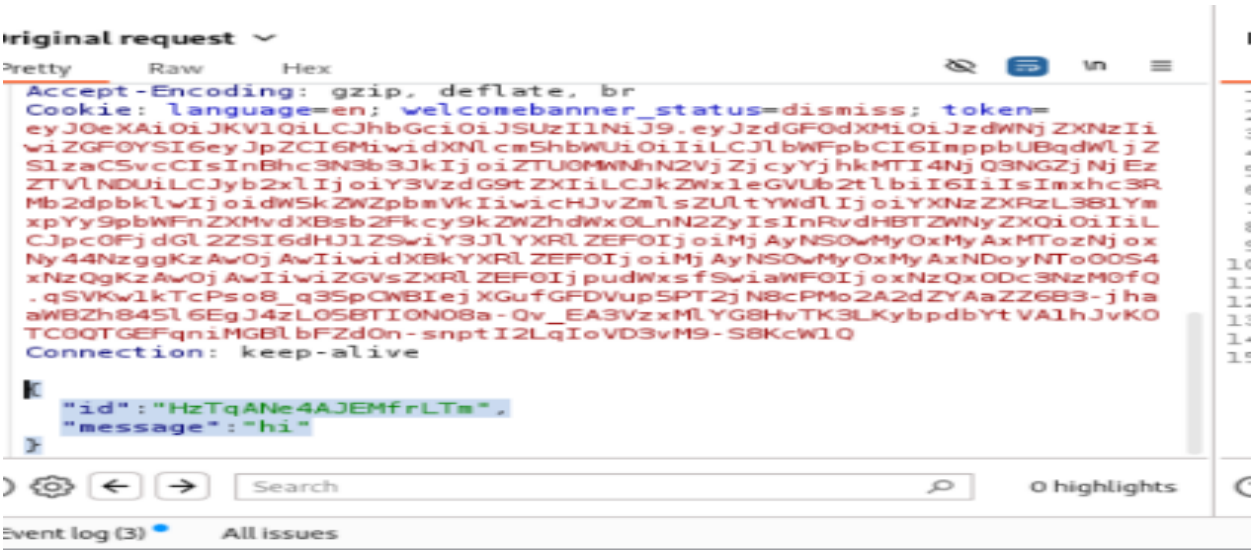


3. NoSQL Injection in Product Reviews

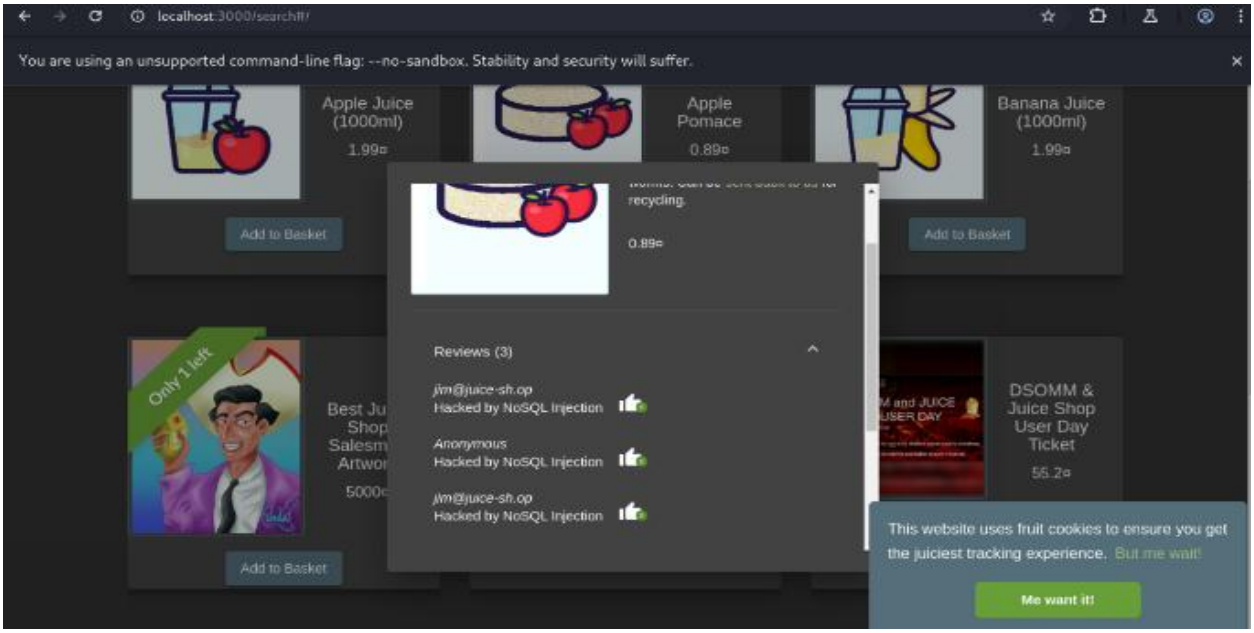
Goal: Manipulate backend queries that use NoSQL (e.g., MongoDB) to alter or retrieve data.

Steps & Clarifications

1. Navigate to a specific product page (e.g., http://localhost:3000/#/product/0).
2. Scroll down to **Reviews**; write a test review (e.g., “hi”).
3. After submission, **click “Edit”** on your review.
4. Intercept the request using **Burp Suite** (or a similar proxy).



5. Look for the id parameter in the HTTP request.
6. Modify it from a normal value to:
7. {"id": {"$ne": "-1"}, "message": "Hacked by NoSQL Injection"}
    - Explanation: {"$ne": "-1"} is MongoDB syntax for “not equal to -1,” which effectively bypasses the normal item-based filter.
8. **Forward** the modified request.
9. **Refresh** the product page to verify if **all reviews** got overwritten with “Hacked by NoSQL Injection.”



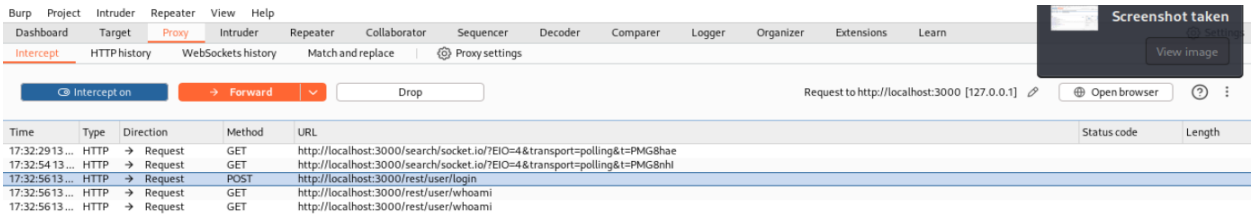
---

4. Password Cracking via Intruder (Burp Suite)

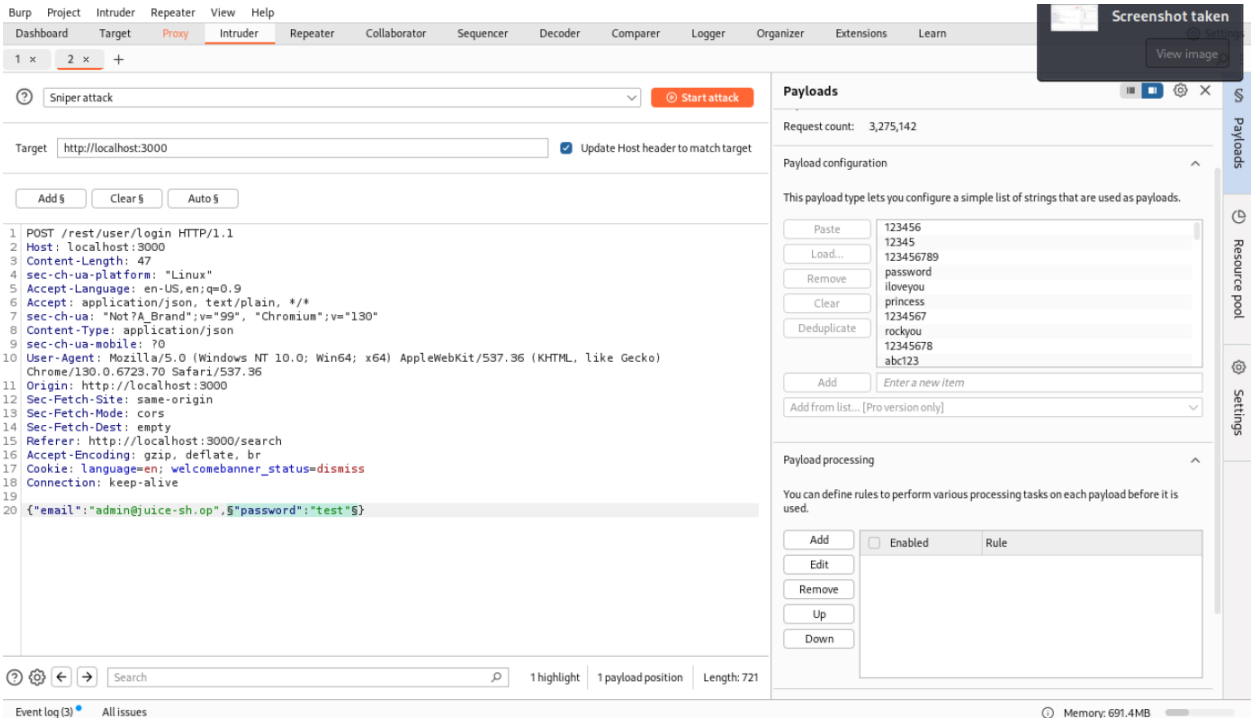
Goal: Use an **Intruder** attack to brute force or guess user credentials.

Steps & Clarifications

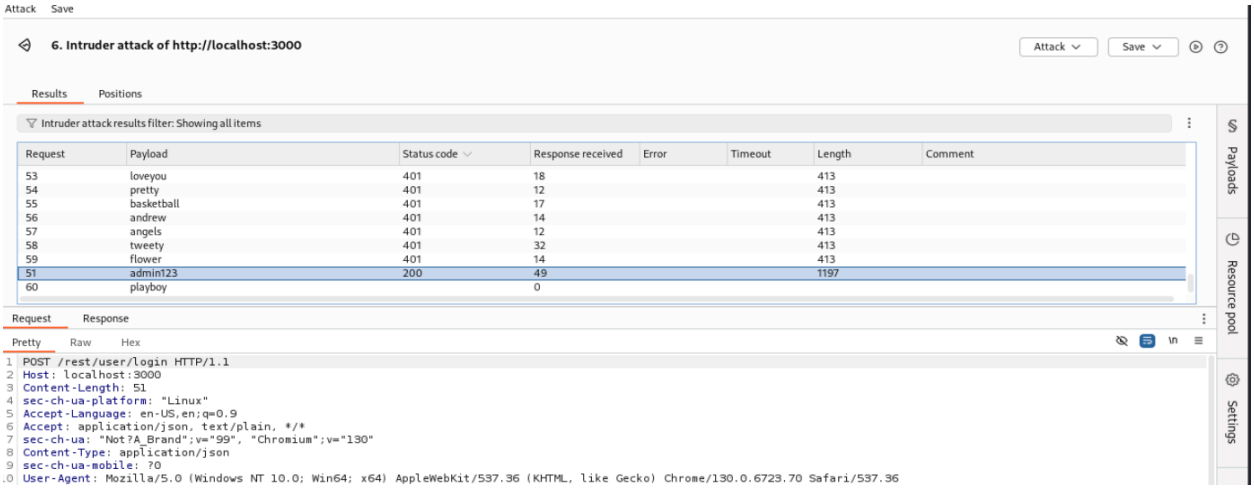
1. **Attempt a normal login** (e.g., with admin@juice-sh.op or some known user).
2. In **Burp Suite**, **send** the login request to **Intruder**.



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



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.

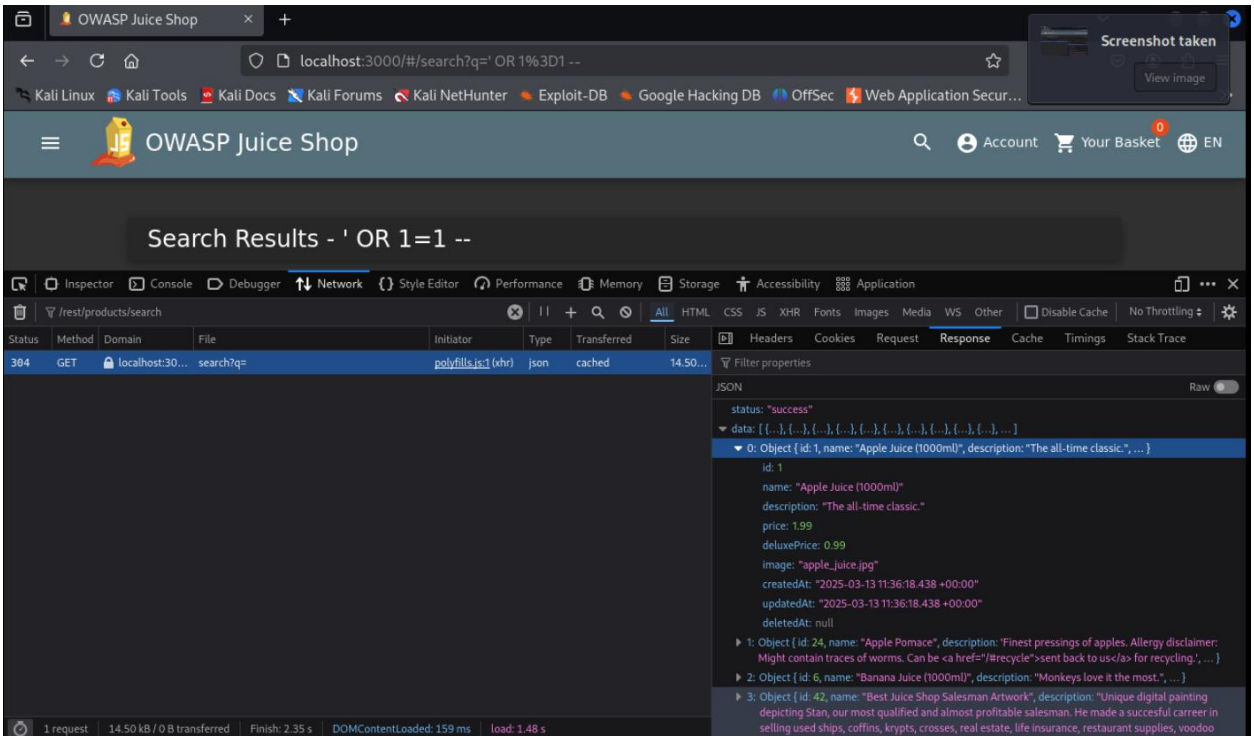


5. Error-Based (Search) SQL Injection

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

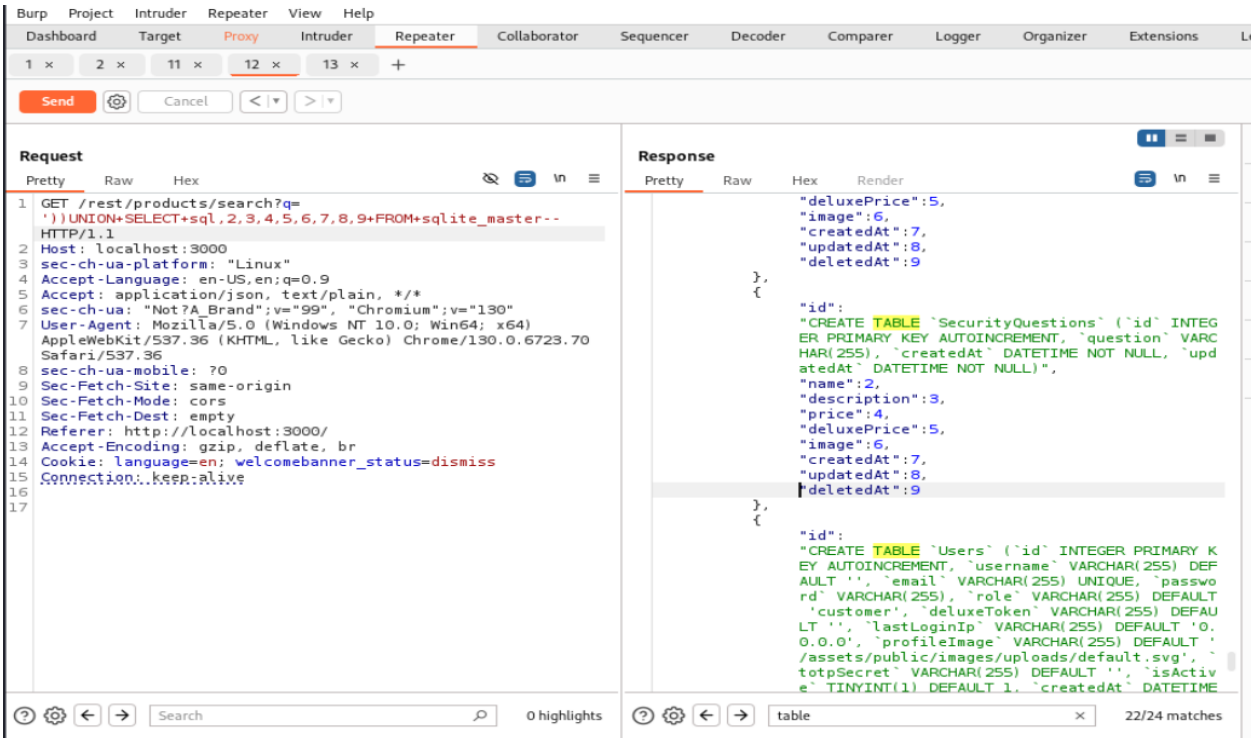
Steps & Clarifications

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.



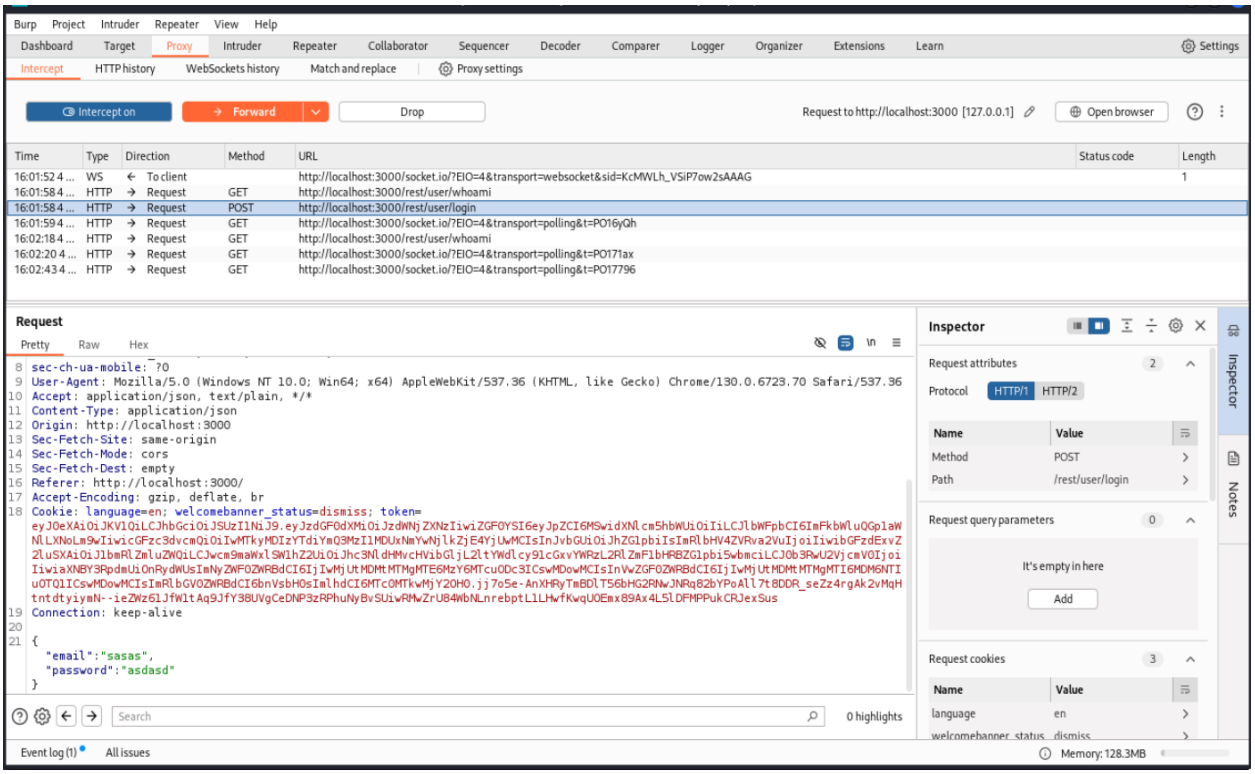
6. Detailed Database Schema Extraction

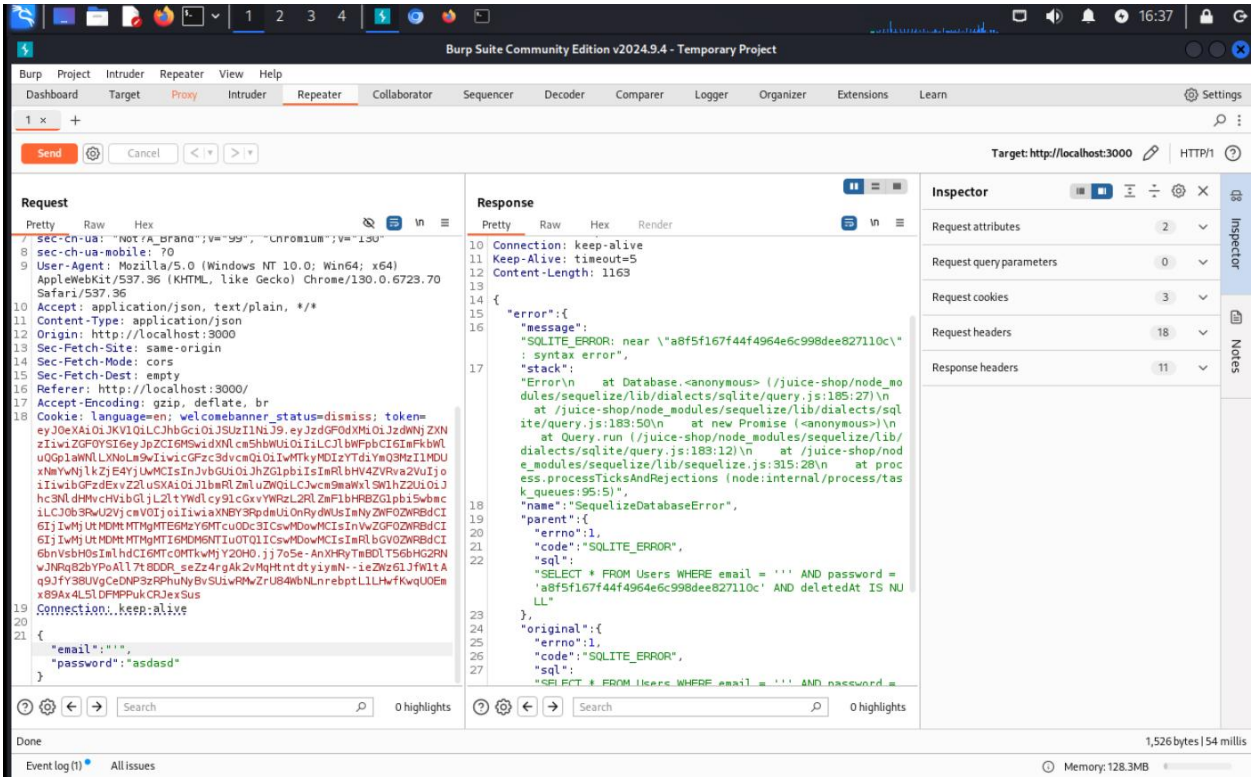
Goal: Understand the database schema to pivot and escalate attacks.



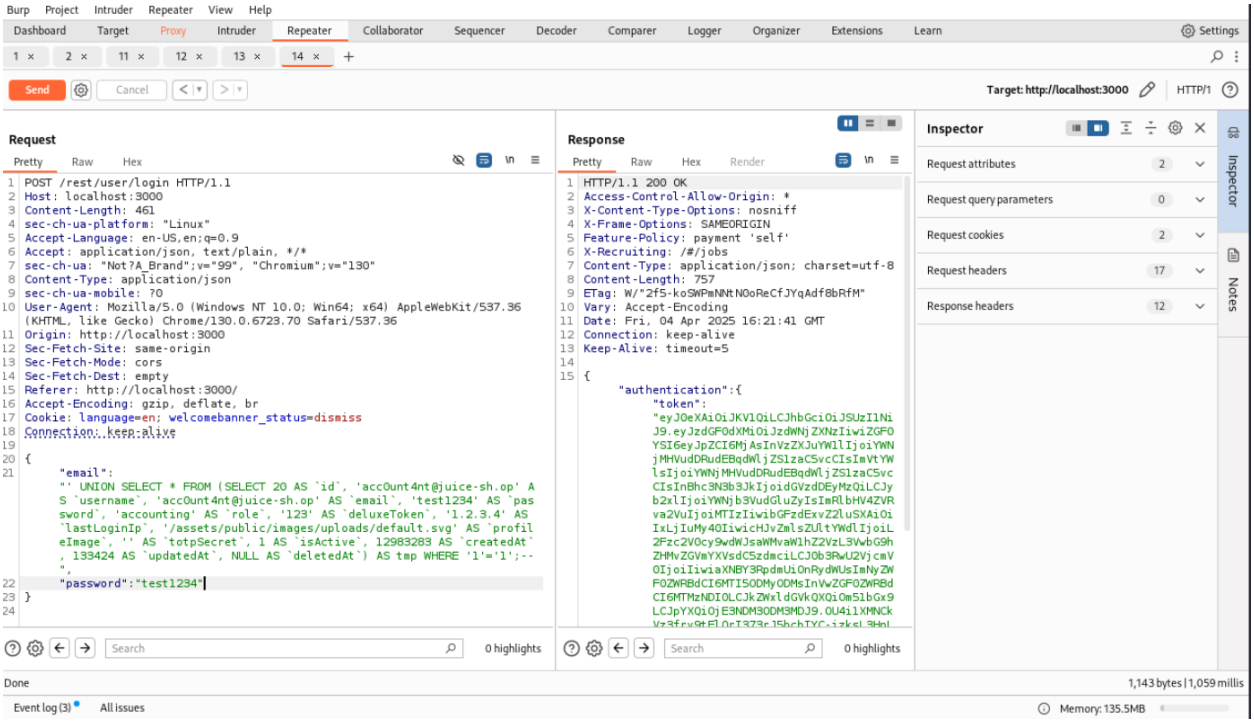
Steps & Clarifications

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





2. Use **UNION SELECT** statements to gather table names, columns, etc.
3. Example injection from the text:
4. test')) UNION SELECT username, password, role, deletedAt, isActive, createdAt, id, email, profileImage FROM USERS--
    - Explanation: This attempts to retrieve columns from the USERS table by combining it with existing queries.

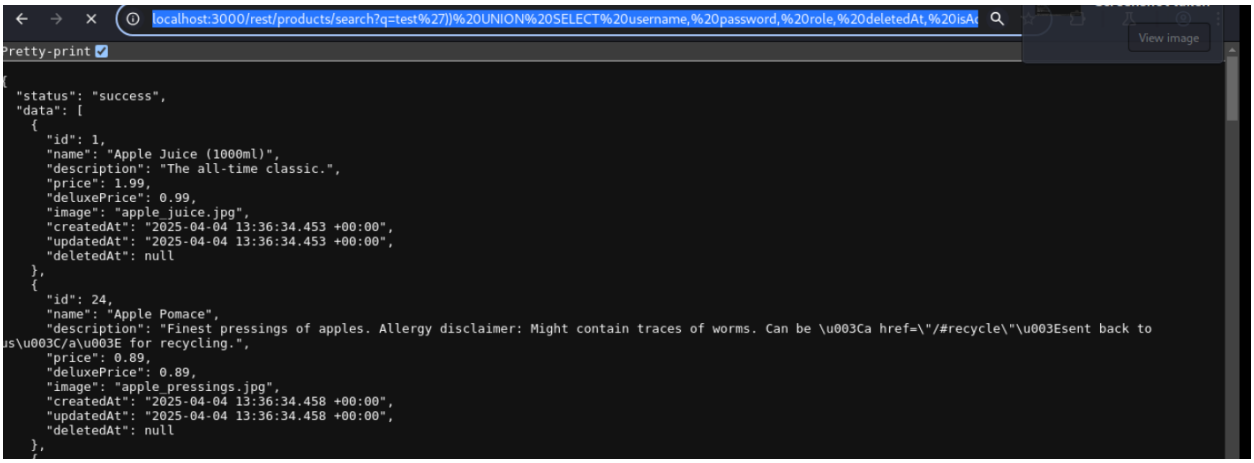


---

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—



Then we got this