TASK 1 – Penetration Testing

Introduction

As part of my penetration testing task, I selected the OWASP Juice Shop application hosted on TryHackMe. This intentionally vulnerable web application provided an excellent platform for identifying and exploiting common web vulnerabilities. The goal of this test was to assess the application's security posture by finding vulnerabilities and demonstrating their potential impact.

Setting up the Environment

I started by accessing the TryHackMe platform and navigating to the Complete Beginner ➔ Web Hacking Fundamentals ➔ OWASP Juice Shop room. This room provided a guided environment with access to the vulnerable Juice Shop web application.

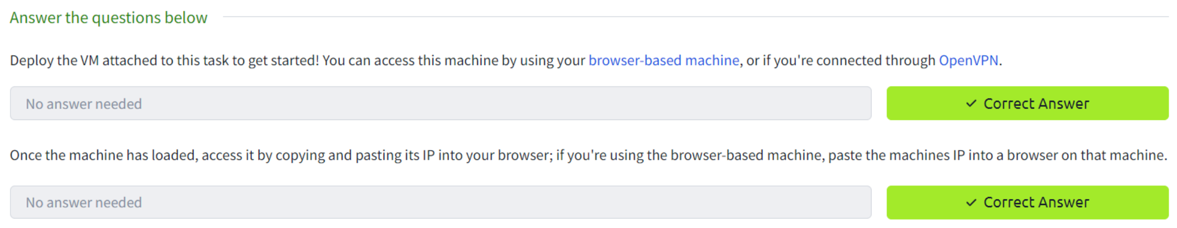
Tools used:

OWASP ZAP: For automated scanning and vulnerability identification.

Burp Suite: For intercepting traffic, manual testing, and exploiting vulnerabilities.

Kali Linux: Pre-installed tools such as curl and terminal commands were also used.

Task 1. Open for business!


Answer the questions below

Deploy the VM attached to this task to get started! You can access this machine by using your browser-based machine, or if you're connected through OpenVPN.

No answer needed

Once the machine has loaded, access it by copying and pasting its IP into your browser; if you're using the browser-based machine, paste the machines IP into a browser on that machine.

No answer needed

Task 2. Let's go on an adventure!

Before diving into the actual hacking, I took some time to explore the site. I opened Burp Suite and set the Intercept mode to off. This allowed me to browse the OWASP Juice Shop freely, while Burp logged all the requests sent to and from the server in the background.

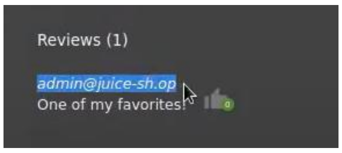
As I navigated through different sections of the shop—searching for products, checking the login page, and viewing individual items—Burp quietly recorded each interaction. I knew these logs could come in handy later when I needed to analyze specific requests or identify vulnerabilities.

By familiarizing myself with the site's functionality and observing how it handled user input, I started identifying potential areas for attack, like form fields, search boxes, and account creation features. This reconnaissance was important for understanding the application and narrowing down likely vulnerability points.

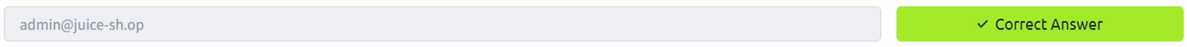
To begin my investigation, I navigated to the Reviews section of the Juice Shop. I knew this might reveal some valuable information about users, including their email addresses.

I clicked on the Apple Juice product, which displayed customer reviews. Among the reviews, I noticed something interesting—the email addresses of the users were visible! Scanning through the list, I quickly identified the Administrator's email address, which was included in one of the reviews.

This provided me with the first important piece of information for further testing.


Reviews (1)

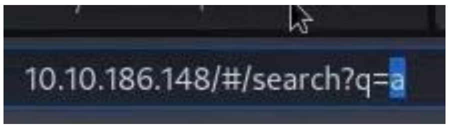
admin@juice-sh.op
One of my favorites!

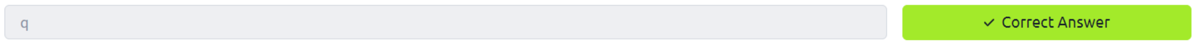

admin@juice-sh.op

Next, I explored the search functionality of the OWASP Juice Shop. By clicking on the magnifying glass in the top-right corner of the application, a search bar appeared. I entered some text and pressed Enter to see how the application handled the search.

As expected, the page reloaded with a new URL reflecting the search query. I noticed that the URL updated to include the text I entered after the /#/search? part. The search parameter was visible, and it was identified by the letter q.

This provided a key insight into how the application processed search queries, which I could use for further testing and potential exploitation.

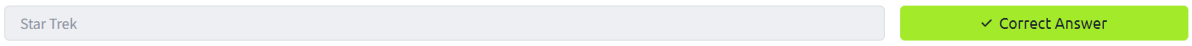

10.10.186.148/#/search?q=a


q

Continuing my exploration, I focused on the reviews section again, specifically looking at the review for the Green Smoothie product. One review caught my attention; it was written by a user named Jim.

In his review, Jim mentioned a replicator. This piqued my curiosity, so I decided to investigate further. A quick Google search for "replicator" yielded results pointing to its origin from the popular TV show Star Trek.

This reference not only added an interesting layer to the reviews but also might be relevant for any challenges or hidden features within the Juice Shop application.

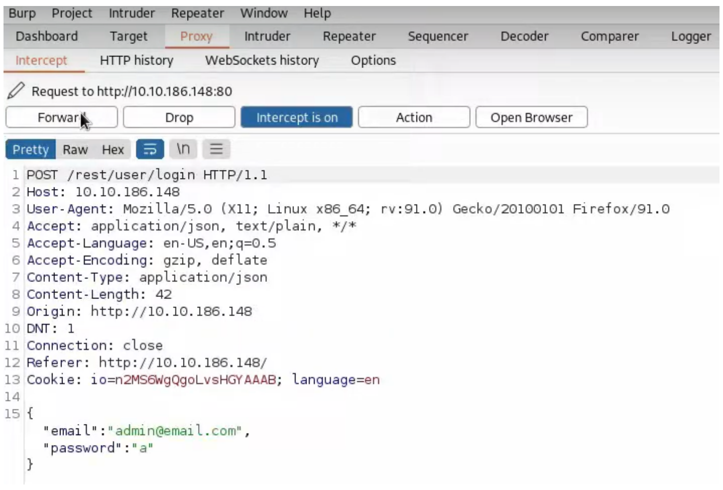

Star Trek

Task 3. Inject the juice

To attempt to log into the administrator account, I first navigated to the login page of the OWASP Juice Shop. Here, I entered some sample data into the email and password fields to prepare for the login attempt.

Before clicking the submit button, I made sure to turn on Intercept mode in Burp Suite. This was crucial because it would allow me to capture the data being sent to the server during the login process.

After clicking submit, Burp intercepted the request. I noticed the email parameter in the request data, so I decided to modify it. Specifically, I changed the value of the email parameter from the sample email to the following injection payload:

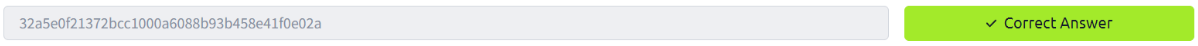

Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger
Intercept HTTP history WebSockets history Options
Request to http://10.10.186.148:80
Forward Drop Intercept is on Action Open Browser
Pretty Raw Hex

```
POST /rest/user/login HTTP/1.1
Host: 10.10.186.148
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Content-Length: 42
Origin: http://10.10.186.148
DNT: 1
Connection: close
Referer: http://10.10.186.148/
Cookie: io=n2MS6WgQgoLvsHGYAAAB; language=en

{
  "email":"admin@email.com",
  "password":"a"
}
```

' or 1=1--

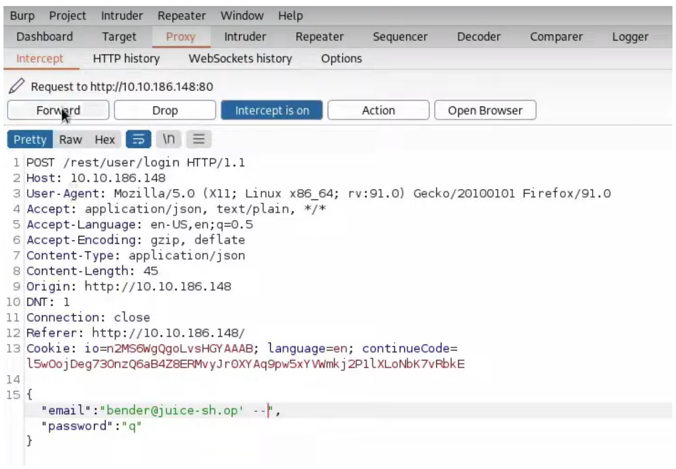
This SQL injection attempt aimed to manipulate the login query to bypass authentication by always returning true. Once I modified it, I forwarded the request to the server to see if it would grant me access to the administrator account.



32a5e0f21372bcc1000a6088b93b458e41f0e02a ✓ Correct Answer

After successfully logging into the administrator account, I proceeded to log into Bender's account using a similar approach. I navigated to the login page again and filled in some random data in the email and password fields.

Once again, I turned Intercept mode on in Burp Suite to capture the login request before sending it to the server. This time, instead of using an SQL injection like ' or 1=1--, I used the following email for the login attempt:


Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger
Intercept HTTP history WebSockets history Options
Request to http://10.10.186.148:80
Forward Drop Intercept is on Action Open Browser
Pretty Raw Hex

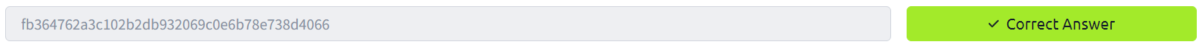
```
POST /rest/user/login HTTP/1.1
Host: 10.10.186.148
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Content-Length: 45
Origin: http://10.10.186.148
DNT: 1
Connection: close
Referer: http://10.10.186.148/
Cookie: io=n2MS6WgQgoLvsHGYAAAB; language=en; continueCode=
l5wOojDeg73OnzQ6aB4Z8ERMvyJrOXYAq9pw5xYVWmkj2P1lXLoNbK7vRbkE

{
  "email":"bender@juice-sh.op' --",
  "password":"q"
}
```

bender@juice-sh.op'--

Since the email address for Bender was valid, there was no need to force the query to return true with 1=1. Instead, I used the -- to comment out the rest of the SQL query, bypassing the need for a valid password.

After making this change, I forwarded the request to the server, and the login was successful—granting me access to Bender's account.



```
fb364762a3c102b2db932069c0e6b78e738d4066
```

Task 4. Who broke my lock?!

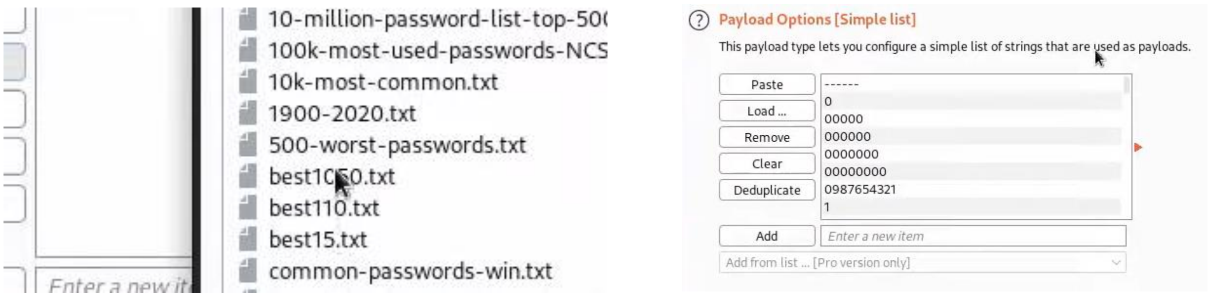
Now that I had previously logged into the Administrator account using SQL Injection, I decided to try brute-forcing the actual password to gain further insight into the security flaw.

First, I captured the login request in Burp Suite again. This time, instead of forwarding it directly to the server, I sent the request to Intruder for a brute-force attack.

Here's how I set it up: I clicked on the Positions tab in Intruder and selected the Clear § button to remove all current attack positions. Then I placed two § symbols inside the quotes of the password field to mark it as the attack point. This tells Burp where to insert the passwords from our wordlist during the brute-force attack.

For the brute-force attack, I used the best1050.txt wordlist from Seclists. To install this on Kali Linux, I ran the following command: apt-get install seclists

Once installed, I loaded the wordlist from the following path: /usr/share/wordlists/SecLists/Passwords/Common-Credentials/best1050.txt



10-million-password-list-top-50
100k-most-used-passwords-NCS
10k-most-common.txt
1900-2020.txt
500-worst-passwords.txt
best1050.txt
best110.txt
best15.txt
common-passwords-win.txt

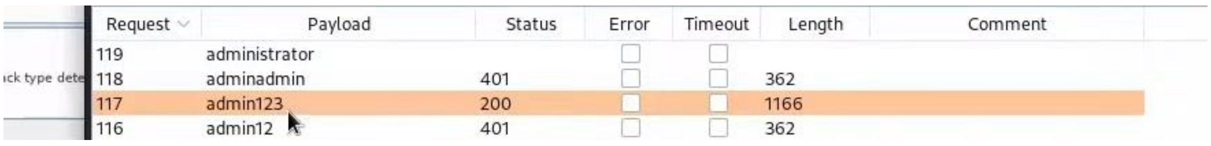
Payload Options [Simple list]

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

```
------
0
00000
000000
0000000
00000000
0987654321
1
```

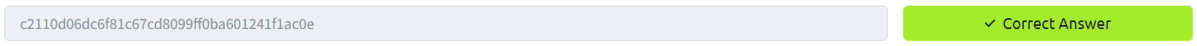
After loading the wordlist into Burp Suite, I started the attack. Burp began trying each password from the list in the password field.

I filtered the results based on the status code. A 401 Unauthorized indicated a failed attempt, while a 200 OK would mean a successful login.



Request	Payload	Status	Error	Timeout	Length	Comment
119	administrator		☐	☐		
118	adminadmin	401	☐	☐	362	
117	admin123	200	☐	☐	1166	
116	admin12	401	☐	☐	362	

After the attack completed, I found a request that returned 200 OK. This meant the correct password had been discovered.

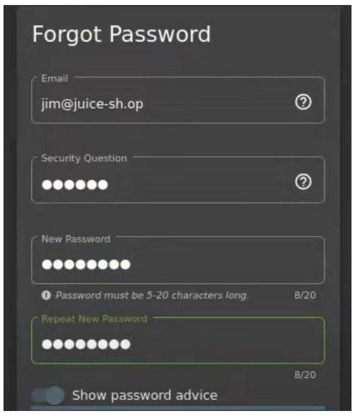


c2110d06dc6f81c67cd8099ff0ba601241f1ac0e

Next, I attempted to exploit the reset password mechanism for Jim's account. I went to the Forgot Password page on the OWASP Juice Shop and entered Jim's email address into the email field. After submitting the request, I was presented with Jim’s security question: "Your eldest sibling’s middle name?"

From my earlier findings, I remembered that Jim had referenced Star Trek in one of his reviews. A quick Google search for "Jim Star Trek" led me to a wiki page for James T. Kirk, a famous character from the series.

After some digging on the wiki page, I discovered that James T. Kirk had a brother named Samuel Kirk. His middle name, Samuel, seemed like the perfect answer to the security question.



Forgot Password

Email: jim@juice-sh.op

Security Question: ●●●●●●

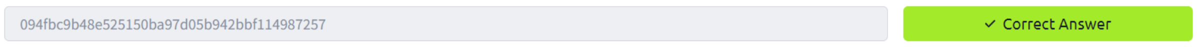
New Password: ●●●●●●●●

Password must be 5-20 characters long. 8/20

Repeat New Password: ●●●●●●●●

8/20

I entered Samuel into the security question field and successfully bypassed the password reset mechanism. This allowed me to change Jim’s password to anything I wanted, granting me full access to his account.



094fbc9b48e525150ba97d05b942bbf114987257

Task 5. AH! Don't look!

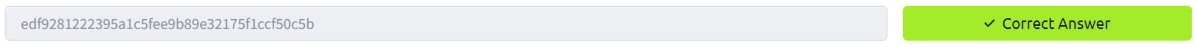
I visited the About Us page on the OWASP Juice Shop and hovered over the link labelled "Check out our terms of use". Upon closer inspection, I noticed that the link pointed to: http://MACHINE_IP/ftp/legal.md

Curious about the directory, I manually navigated to the /ftp/ directory by entering: http://MACHINE_IP/ftp/

To my surprise, the directory was publicly accessible, revealing several files of interest, including acquisitions.md.

I proceeded to download the acquisitions.md file, which contained confidential information. I saved this file for further review.

After successfully downloading the document, I navigated back to the home page of the OWASP Juice Shop. Upon returning, I received a flag confirming that the confidential document had been accessed.

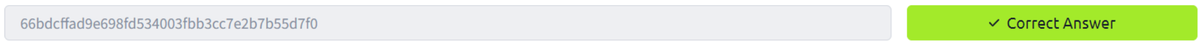


edf9281222395a1c5fee9b89e32175f1ccf50c5b

While watching the video, I paid close attention to certain parts of the song that stood out. MC SafeSearch mentioned that his password was "Mr. Noodles", but he had replaced some vowels with zeros. Specifically, he indicated that he had replaced the letter "o" with the number "0".

Based on this hint, I deduced that the password for the mc.safesearch@juice-sh.op account was "Mr. N00dles", with the "o"s in "Noodles" replaced by zeros.

I used mc.safesearch@juice-sh.op as the email and Mr. N00dles as the password. Upon submitting the credentials, I successfully logged into MC SafeSearch's account.



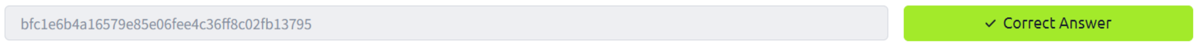
66bdcffad9e698fd534003fbb3cc7e2b7b55d7f0

I went back to the /ftp/ directory to download the package.json.bak file. However, upon trying to download the file directly, I was met with a 403 Forbidden error, indicating that only .md and .pdf files were allowed for download.

To bypass this restriction, I employed a technique called the Poison Null Byte. A Null Byte is represented as %00, but to use it in a URL, I needed to URL-encode it, making it appear as %2500. By appending this to the filename, followed by .md, I tricked the server into thinking it was allowing a .md file download.

http://MACHINE_IP/ftp/package.json.bak ➔
http://MACHINE_IP/ftp/package.json.bak%2500.md

This Poison Null Byte caused the server to ignore the .bak extension and treat the file as a .md file, bypassing the 403 error. I was then able to successfully download the package.json.bak file.



bfc1e6b4a16579e85e06fee4c36ff8c02fb13795

Task 6. Who's flying this thing?

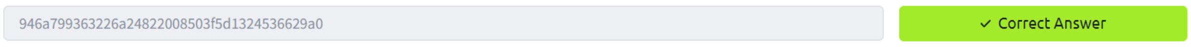
I started by opening the Debugger in Firefox by navigating to the Web Developers menu. This gave me access to the JavaScript files loaded on the page. I refreshed the page and searched for the main-es2015.js file, which contains critical scripts for the application. I then visited the URL: http://MACHINE_IP/main-es2015.js

To make the JavaScript code more readable, I clicked the { } button at the bottom, which prettified the code. This allowed me to search through it more easily.

I searched the file for the term "admin". After browsing through several results, I found a line of code that pointed to "path: administration". A few lines below this, I found a reference to a page with the path "/#/administration", which indicated the existence of an administration page.

I tried to access the "/#/administration" page, but since I wasn’t logged into an administrator account, I couldn’t view it. This confirmed that access to the page was restricted to admin users.

After analyzing the main-es2015.js file and identifying the administration page path, I logged out and logged back in using the admin credentials we previously obtained. Then, I manually added /#/administration to the URL. This allowed me to access the administration page, where I successfully found the flag.

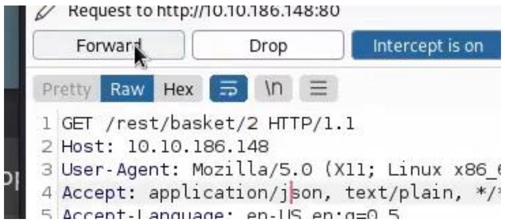


946a799363226a24822008503f5d1324536629a0

I logged into the Admin account and navigated to the 'Your Basket' section. At the same time, I ensured that Burp Suite was running to capture the HTTP requests being sent from the browser to the server.

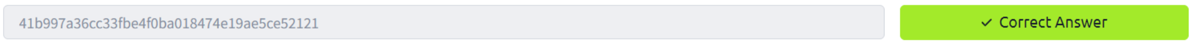
While clicking around the Your Basket section, Burp Suite captured the request.

I noticed that the number 1 after /basket/ represented the basket for the current user. To view another user's basket, I changed the number 1 to 2.



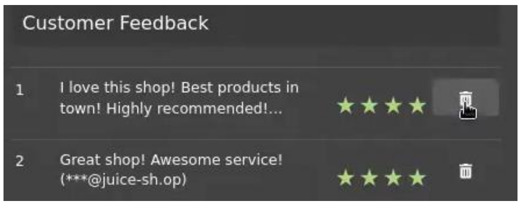
```
GET /rest/basket/2 HTTP/1.1
Host: 10.10.186.148
User-Agent: Mozilla/5.0 (X11; Linux x86_
Accept: application/json, text/plain, */
Accept-Language: en-US,en;q=0.5
```

After forwarding this modified request to the server, I was able to see the shopping basket for UserID 2. This technique could be repeated to view other users' baskets, provided they have one.



41b997a36cc33fbe4f0ba018474e19ae5ce52121

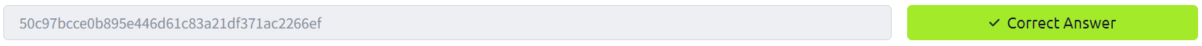
I logged into the admin account and navigated to the administration page. On the administration page, I scrolled through the list of reviews and located the ones that had a 5-star rating. To remove the 5-star reviews, I simply clicked the bin icon next to each review with a 5-star rating. Each review was successfully deleted from the system.



Customer Feedback

1 I love this shop! Best products in town! Highly recommended!... ★★★★

2 Great shop! Awesome service! (***@juice-sh.op) ★★★★

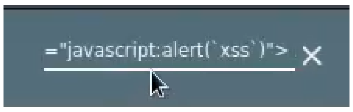


50c97bcce0b895e446d61c83a21df371ac2266ef

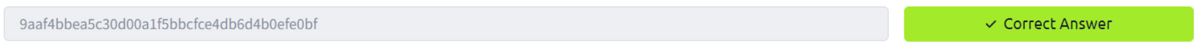
Task 7. Where did that come from?

I used the following iframe code, which includes a JavaScript alert: <iframe src="javascript:alert(`xss`)">

I entered this code directly into the search bar of the OWASP Juice Shop application. Once I hit Enter, the search bar processed the input, and the alert box was triggered, confirming the presence of a DOM-based XSS vulnerability.



```
="javascript:alert(`xss`)">
```

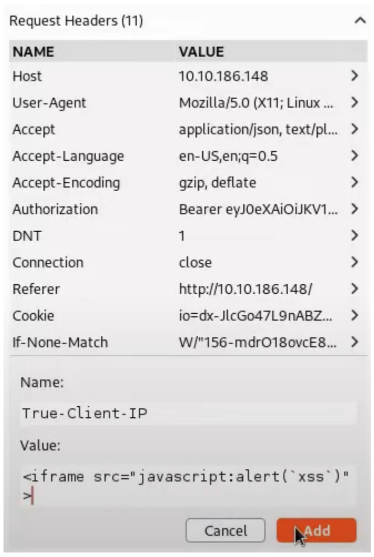


9aaf4bbea5c30d00a1f5bbcfce4db6d4b0efe0bf

I first logged into the admin account and navigated to the Last Login IP page. This page displayed the last recorded IP address, which was either 0.0.0.0 or something in the 10.x.x.x range.

To log a new IP address, I logged out of the account while making sure Burp Suite was intercepting the request. This allowed me to capture the logout request and modify it.

In Burp Suite, I navigated to the Headers tab of the intercepted request. Here, I added the following custom header to inject the XSS payload ➔ True-Client-IP: <iframe src="javascript:alert(`xss`)">



Request Headers (11)

NAME	VALUE
Host	10.10.186.148
User-Agent	Mozilla/5.0 (X11; Linux ...
Accept	application/json, text/pl...
Accept-Language	en-US,en;q=0.5
Accept-Encoding	gzip, deflate
Authorization	Bearer eyJ0eXAiOiJKV1...
DNT	1
Connection	close
Referer	http://10.10.186.148/
Cookie	io=dx-JlcGo47L9nABZ...
If-None-Match	W/"156-mdrO18ovcE8...

Name:

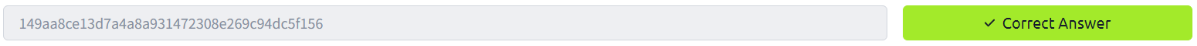
```
True-Client-IP
```

Value:

```
<iframe src="javascript:alert(`xss`)"
>
```

After adding the malicious True-Client-IP header, I forwarded the request to the server.

Upon logging back into the admin account and navigating to the Last Login IP page, the XSS attack was triggered, showing the alert box with the "xss" message. This confirmed that the application was vulnerable to persistent XSS, as the malicious payload persisted and was executed when the page loaded.



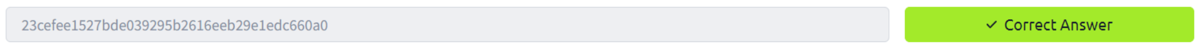
149aa8ce13d7a4a8a931472308e269c94dc5f156

First, I logged into the admin account and navigated to the Order History page. On this page, there is a "Truck" icon associated with each order. Clicking this icon brings up the track result page, which contains a tracking ID for the order.

I noticed that the ID parameter in the URL looked like 5267-f73dcd000abcc353.

To perform the reflected XSS attack, I replaced the tracking ID with the following XSS payload ➔ <iframe src="javascript:alert(`xss`)">

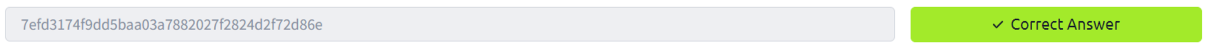
After modifying the URL with the XSS payload, I refreshed the page. The alert box with the message "xss" popped up, confirming the successful execution of a reflected XSS attack.



23cefee1527bde039295b2616eeb29e1edc660a0

Task 8. Exploration!

I entered the following URL to access the scoreboard page ➔ .../#/score-board/ page



7efd3174f9dd5baa03a7882027f2824d2f72d86e