COURSE: CLOUD AND NETWORK SECURITY

NAME: DENISE SOPHY ONDISO MUTAYI

STUDENT NO: CS-CN09-25047

WEB REQUESTS

Table of Contents

Introduction

This lab focused on analyzing HTTP and HTTPS traffic using tools such as Wireshark and curl to understand request/response mechanics, status codes, and protocol behaviors. Through hands-on inspection of GET requests, headers, status codes, cookies, and secure HTTPS sessions, the exercise provided insight into how web clients and servers interact, as well as how to dissect this communication at the packet level. This understanding is critical for network analysts, cybersecurity professionals, and web developers alike, offering visibility into the inner workings of everyday web interactions.

HyperText Transfer Protocol (HTTP)

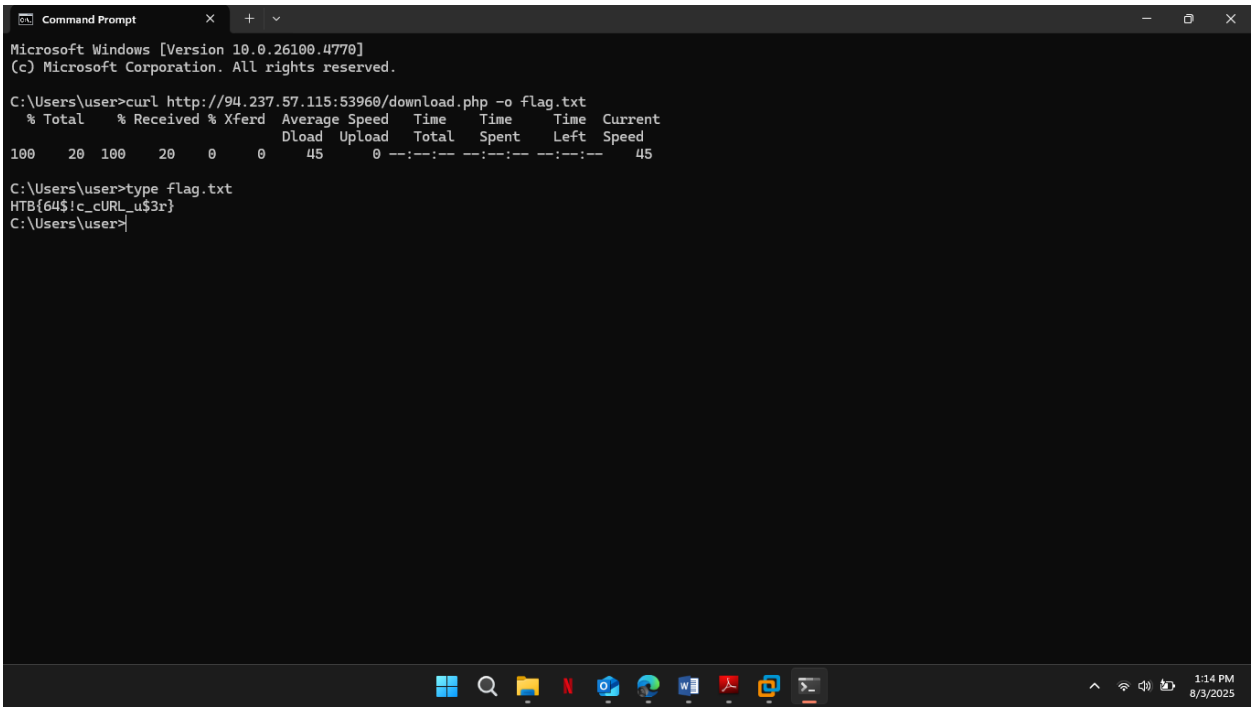
HTTP is an application-level protocol used to access the World Wide Web resources.

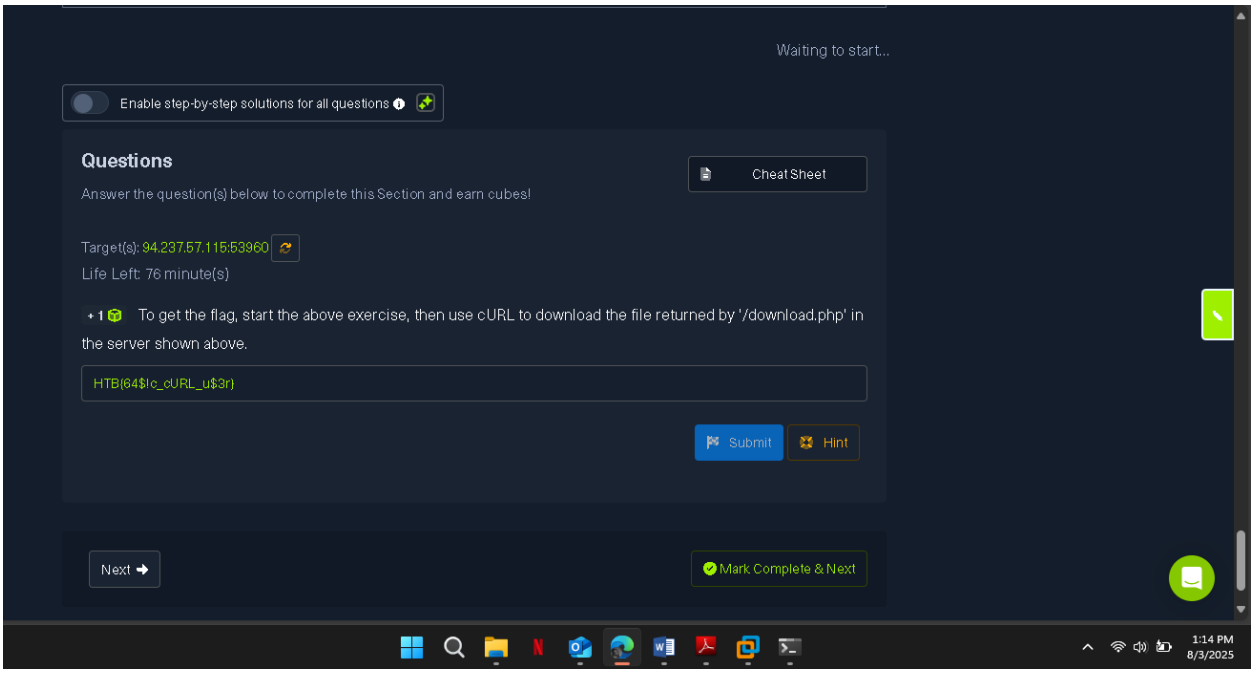
cURL Command-Line Tool

cURL is a versatile command-line utility used to transfer data over various protocols, with strong support for HTTP. It is widely used in web penetration testing and automation for sending HTTP requests without rendering web content like browsers do.

Basic usage involves providing a URL to retrieve the raw HTTP response. Additional flags allow for enhanced functionality such as saving output to files (-O or -o), silencing status output (-s), sending POST data (-d), including response headers (-i), and specifying authentication credentials (-u).

The tool's flexibility makes it essential for testing, scripting, and debugging HTTP interactions in professional environments.



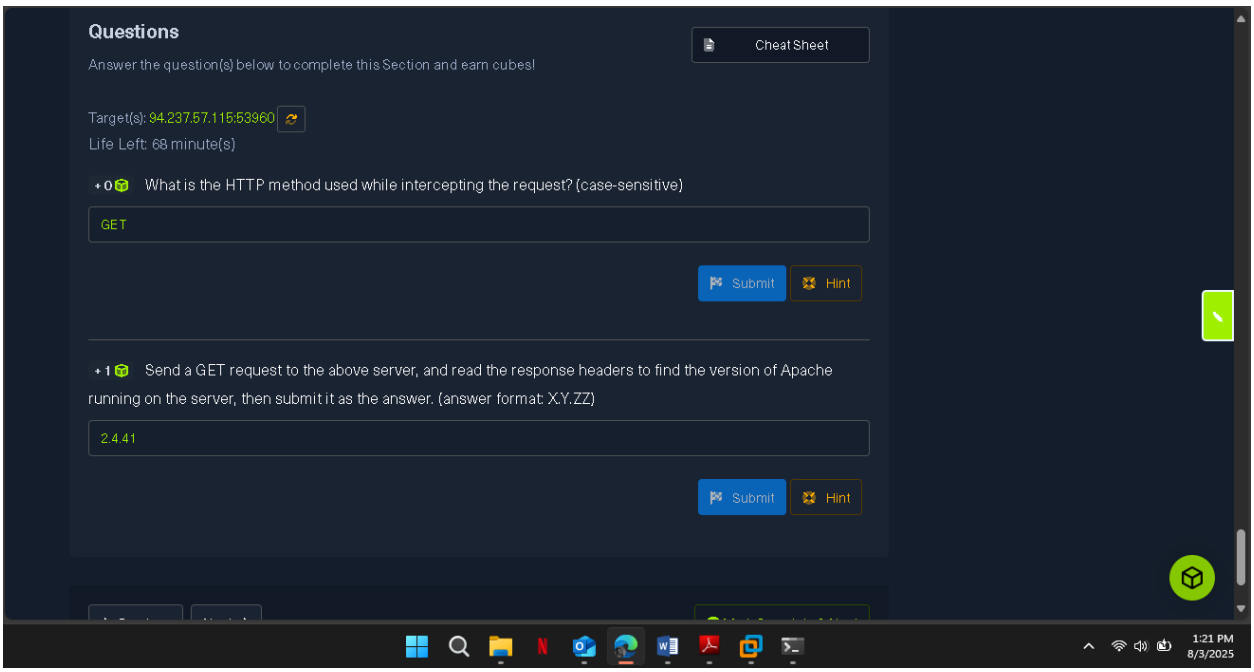


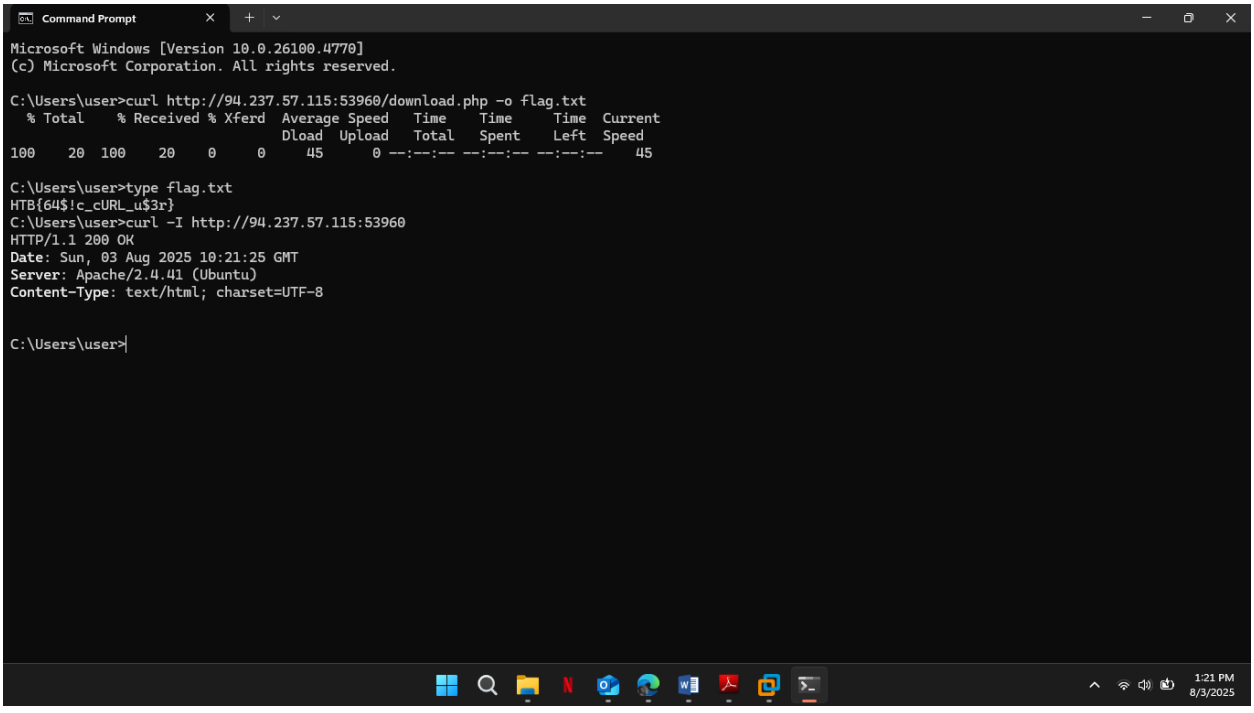
Hypertext Transfer Protocol Secure (HTTPS)

HTTPS is the secure evolution of HTTP, designed to protect data transmitted over the internet by encrypting communication between clients (browsers) and servers. Unlike HTTP, which sends data in plain text vulnerable to interception (e.g., exposing login credentials during transmission), HTTPS uses encryption protocols (TLS/SSL) to prevent man-in-the-middle (MitM) attacks, ensuring that sensitive information remains confidential even if intercepted.

HTTP REQUESTS AND RESPONSES

An HTTP request is made by the client (e.g. cURL/browser), and is processed by the server (e.g. web server). The requests contain all of the details we require from the server, including the resource (e.g. URL, path, parameters)


Questions
Answer the question(s) below to complete this Section and earn cubes!
Cheat Sheet
Target(s): 94.237.57.115:53960
Life Left: 68 minute(s)
+0 What is the HTTP method used while intercepting the request? (case-sensitive)
GET
Submit
Hint
+1 Send a GET request to the above server, and read the response headers to find the version of Apache running on the server, then submit it as the answer. (answer format: X.Y.ZZ)
2.4.41
Submit
Hint
1:21 PM
8/3/2025


Command Prompt
Microsoft Windows [Version 10.0.26100.4770]
(c) Microsoft Corporation. All rights reserved.
C:\Users\user>curl http://94.237.57.115:53960/download.php -o flag.txt
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 20 100 20 0 0 45 0 --:--:-- --:--:-- --:--:-- 45
C:\Users\user>type flag.txt
HTB{64$!c_cURL_u$3r}
C:\Users\user>curl -I http://94.237.57.115:53960
HTTP/1.1 200 OK
Date: Sun, 03 Aug 2025 10:21:25 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Type: text/html; charset=UTF-8
C:\Users\user>
1:21 PM
8/3/2025

HTTP Headers, cURL Usage, and Header Analysis Tools

1. HTTP Header Categories

HTTP headers are categorized based on their context and usage within the request-response lifecycle. The key categories include:

a. General Headers

These are applicable to both requests and responses and provide general information about the communication.

- **Date**: Indicates the timestamp of the message.
- **Connection**: Controls whether the network connection remains open (keep-alive) or is closed (close) after the transaction.

b. Entity Headers

These headers describe the content being transmitted (the entity body) and can appear in both requests and responses.

- **Content-Type**: Specifies the MIME type (e.g., text/html, application/json).
- **Content-Length**: Indicates the size of the message body in bytes.
- **Content-Encoding**: Describes any encoding transformations (e.g., gzip).
- **Boundary**: Used to separate multipart message components.

c. Request Headers

Sent by the client, these headers define request metadata and client capabilities.

- **Host**: Identifies the destination domain or IP address.
- **User-Agent**: Describes the client software initiating the request.
- **Referer**: Indicates the source page that linked to the resource.
- **Accept**: Lists acceptable media types.
- **Cookie**: Includes session identifiers and client-specific data.
- **Authorization**: Used for client authentication via schemes like Basic or Bearer tokens.

d. Response Headers

These are generated by the server to describe the response or provide directives.

- **Server**: Provides information about the server software.
- **Set-Cookie**: Instructs the client to store cookies.

- **WWW-Authenticate**: Requests authentication credentials for protected resources.

e. Security Headers

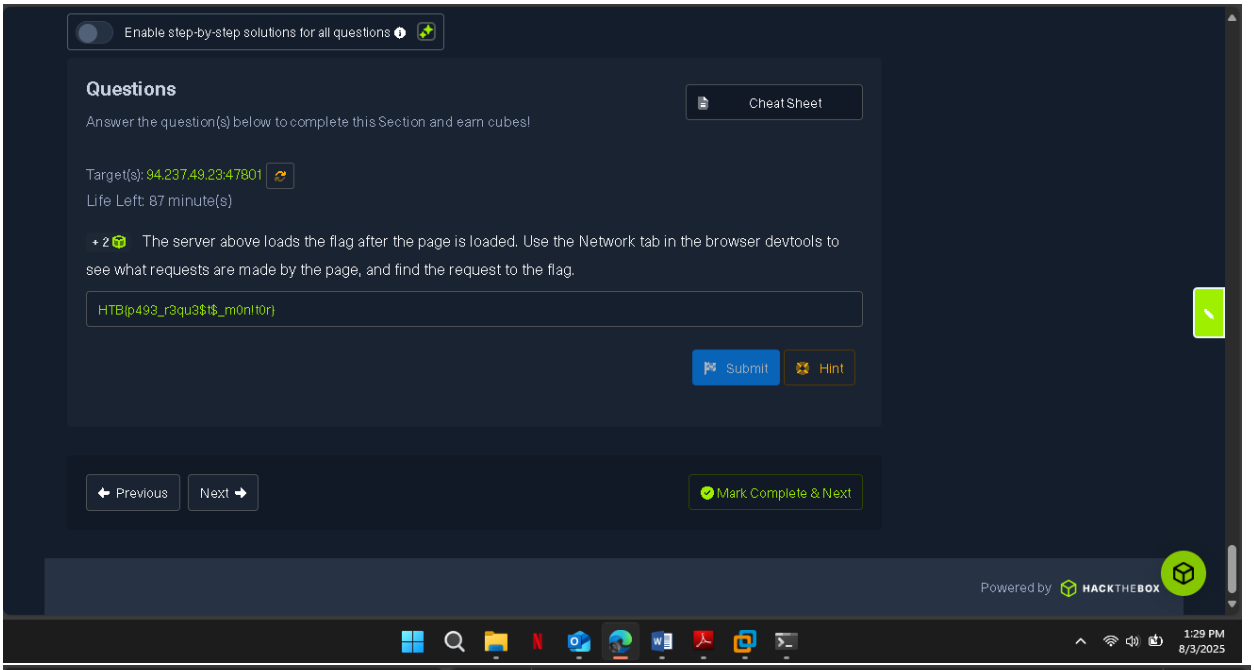
Security headers are sent in server responses to enforce secure behavior in clients, primarily browsers.

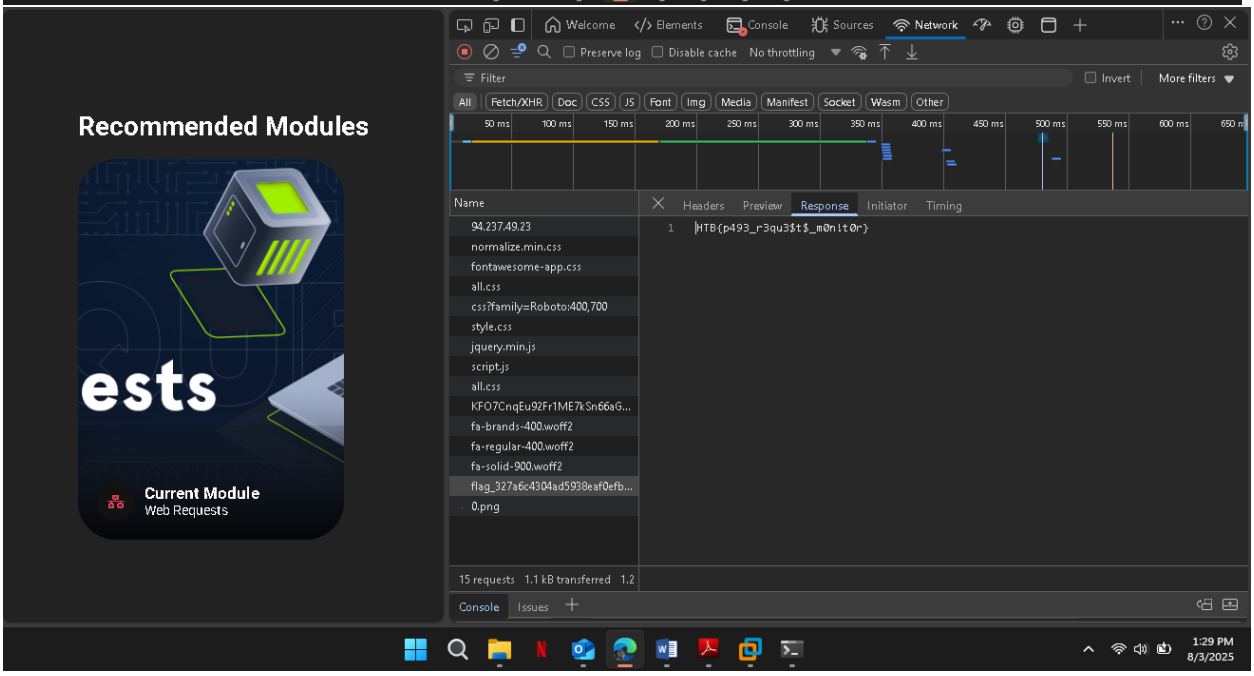
- **Content-Security-Policy (CSP)**: Restricts external resource loading to mitigate XSS attacks.
- **Strict-Transport-Security (HSTS)**: Enforces the use of HTTPS.
- **Referrer-Policy**: Controls the inclusion of referrer data in requests.

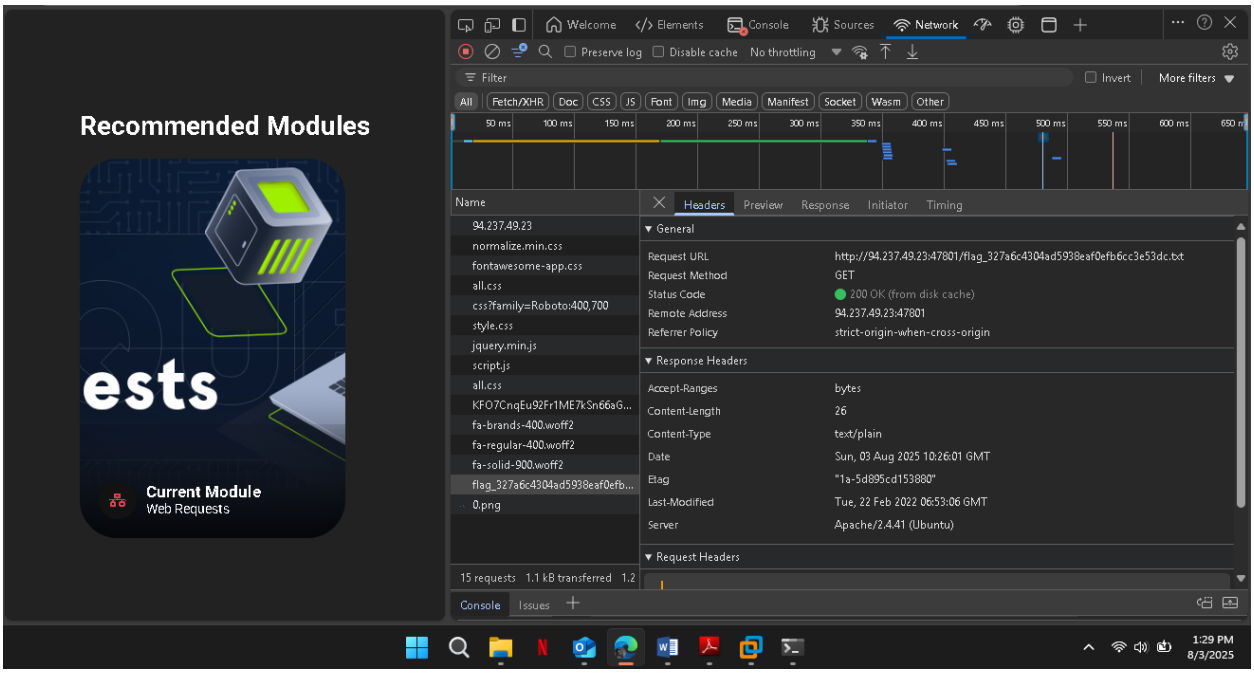
2. Using cURL for Header Inspection

The curl command-line utility allows direct interaction with HTTP servers and facilitates header analysis.

- curl -I <url>: Sends a HEAD request to retrieve response headers only.
- curl -v <url>: Provides a verbose output showing both request and response headers.
- curl -i <url>: Displays the full response including headers and body.
- curl -A <agent>: Sets a custom User-Agent string.
- curl -H "<Header>: <Value>": Adds arbitrary headers to the request.


Enable step-by-step solutions for all questions
Questions
Answer the question(s) below to complete this Section and earn cubes!
Cheat Sheet
Target(s): 94.237.49.23:47801
Life Left: 87 minute(s)
+ 2 The server above loads the flag after the page is loaded. Use the Network tab in the browser devtools to see what requests are made by the page, and find the request to the flag.
HTB{p493_r3qu3t_m0nit0r}
Submit
Hint
Previous
Next
Mark Complete & Next
Powered by HACKTHEBOX
1:29 PM
8/3/2025


Recommended Modules
Current Module
Web Requests
Welcome
Elements
Console
Sources
Network
Preserve log
Disable cache
No throttling
Filter
Invert
More filters
All
Fetch/XHR
Doc
CSS
JS
Font
Img
Media
Manifest
Socket
Wasm
Other
Name
Headers
Preview
Response
Initiator
Timing
94.237.49.23
normalize.min.css
fontawesome-app.css
all.css
css?family=Roboto:400,700
style.css
jquery.min.js
script.js
all.css
fa-brands-400.woff2
fa-regular-400.woff2
fa-solid-900.woff2
flag_327a6c4304ad5938eaf0efb...
0.png
1 HTB{p493_r3qu3t_m0nit0r}
15 requests 1.1 kB transferred 1.2
Console
Issues
1:29 PM
8/3/2025



HTTP GET Requests and Basic Authentication

In web security assessments, comprehending the behavior of HTTP requests is vital for analyzing how web applications communicate with their backends. This report explores the practical use of browser developer tools, command-line tools like cURL, and HTTP headers to dissect **GET requests** and **Basic Authentication** mechanisms within a target web application.

1. Observing HTTP GET Requests via Browser DevTools

Upon visiting a website, the browser issues an initial GET request to fetch the main page. Subsequent resource fetches—such as scripts, stylesheets, or data—can also be monitored using the **Network tab** in browser dev tools. This monitoring is invaluable for penetration testers and bug bounty hunters, offering a real-time view of how frontends interact with backends.

2. HTTP Basic Authentication

Unlike modern login forms that use POST methods, **Basic HTTP Authentication** prompts the browser itself to handle credential submission. In this exercise:

- The protected page was accessed via http://<SERVER_IP>:<PORT>/, triggering a 401 Unauthorized response.
- Authentication credentials (admin:admin) were transmitted either:
    - via cURL using -u admin:admin
    - via the URL itself: http://admin:admin@<SERVER_IP>:<PORT>/
    - or manually through the Authorization header using -H 'Authorization: Basic <base64>'

These methods successfully authenticated the session, proving that the server relied on standard HTTP headers for user validation.

3. Manual Header Manipulation and Response Analysis

Using cURL with the -v flag showed how credentials are encoded in base64 (admin:admin → YWRtaW46YWRtaW4=) and embedded in the request header as:

4. URL Parameters and GET Requests

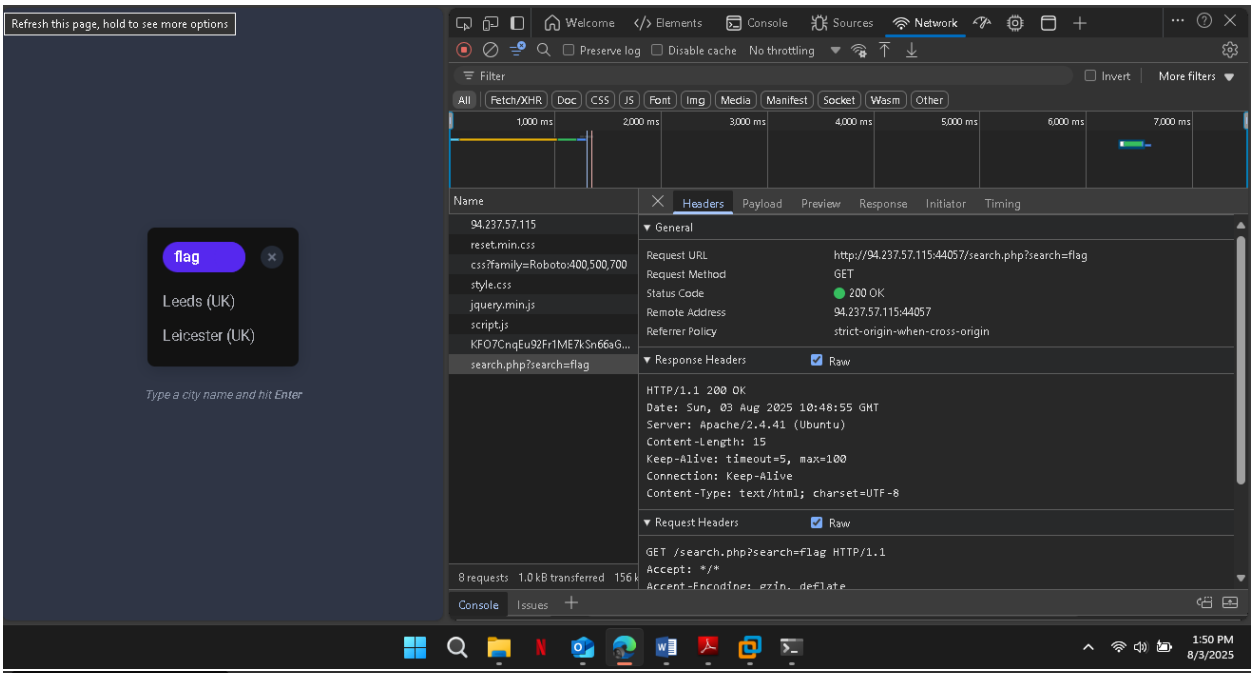
Once authenticated, the application revealed a **city search feature** that leveraged GET parameters to retrieve dynamic content. Monitoring the Network tab during a search operation (e.g., for “le”) revealed that the frontend requested a resource like:

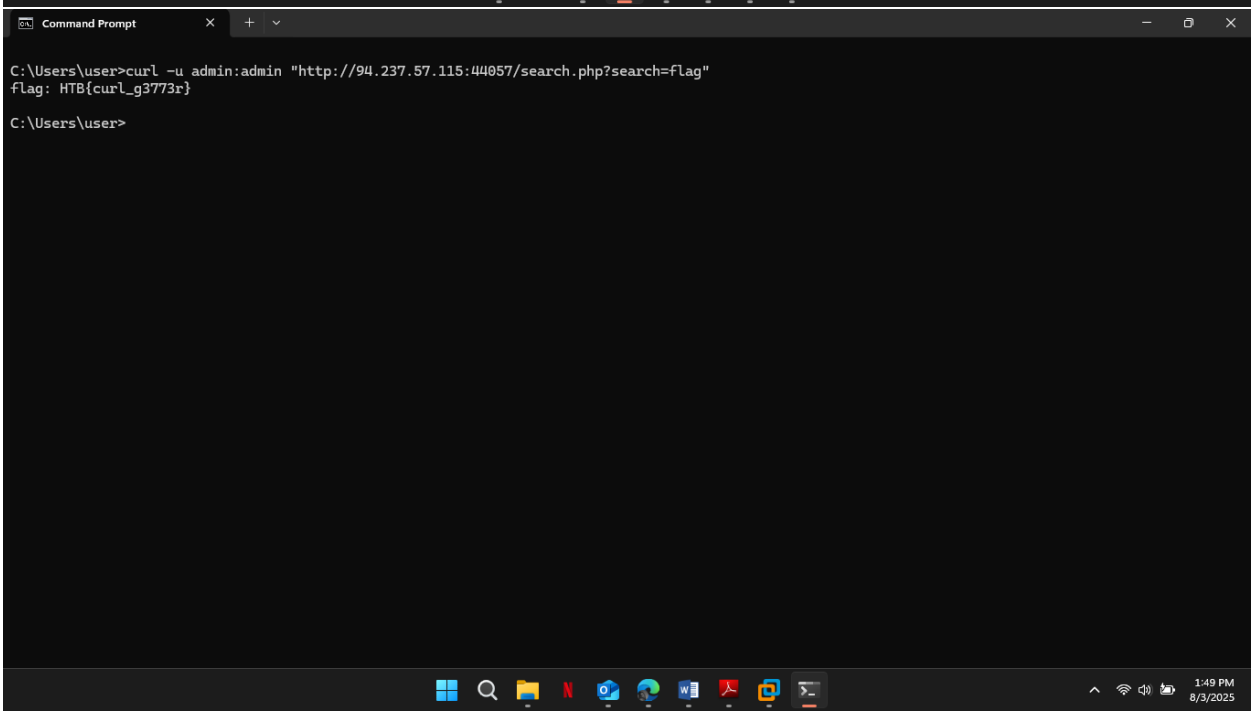
5. Browser Copy Features for Automation

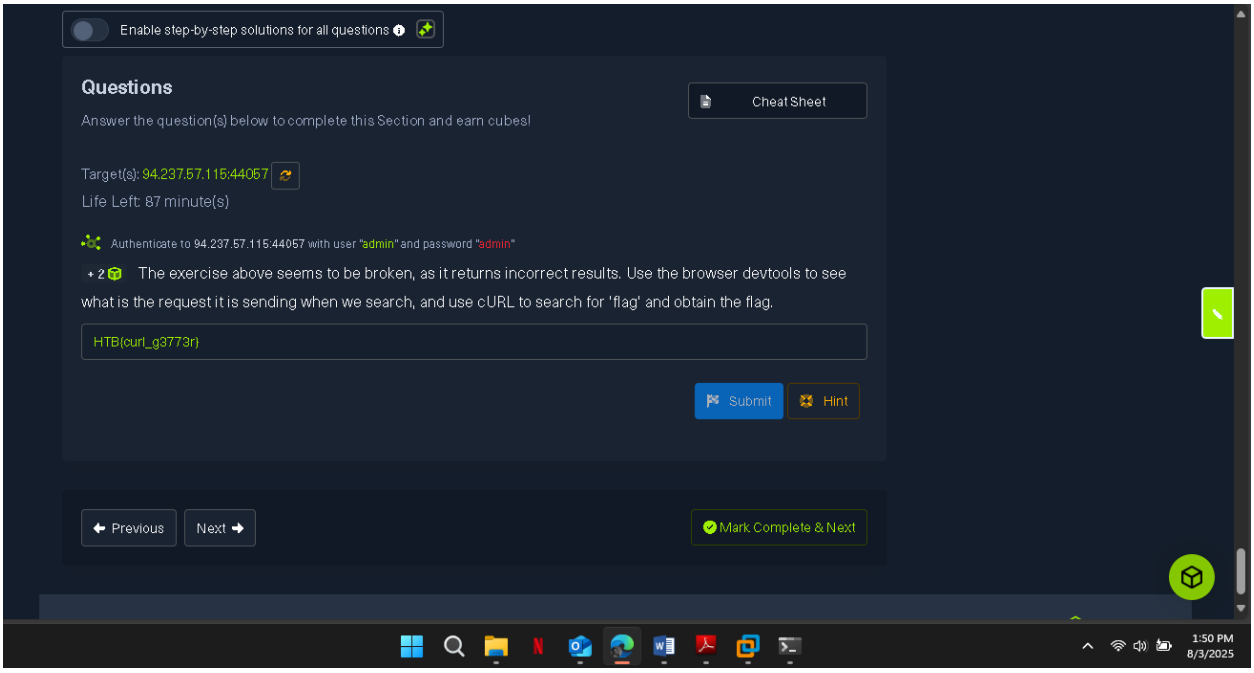
DevTools simplify automation by offering:

- **Copy as cURL**: Replicates exact HTTP requests for CLI execution.
- **Copy as Fetch**: Generates equivalent JavaScript code using the Fetch API for console-based testing.

These features accelerate reconnaissance, debugging, and scripting during manual web application testing.

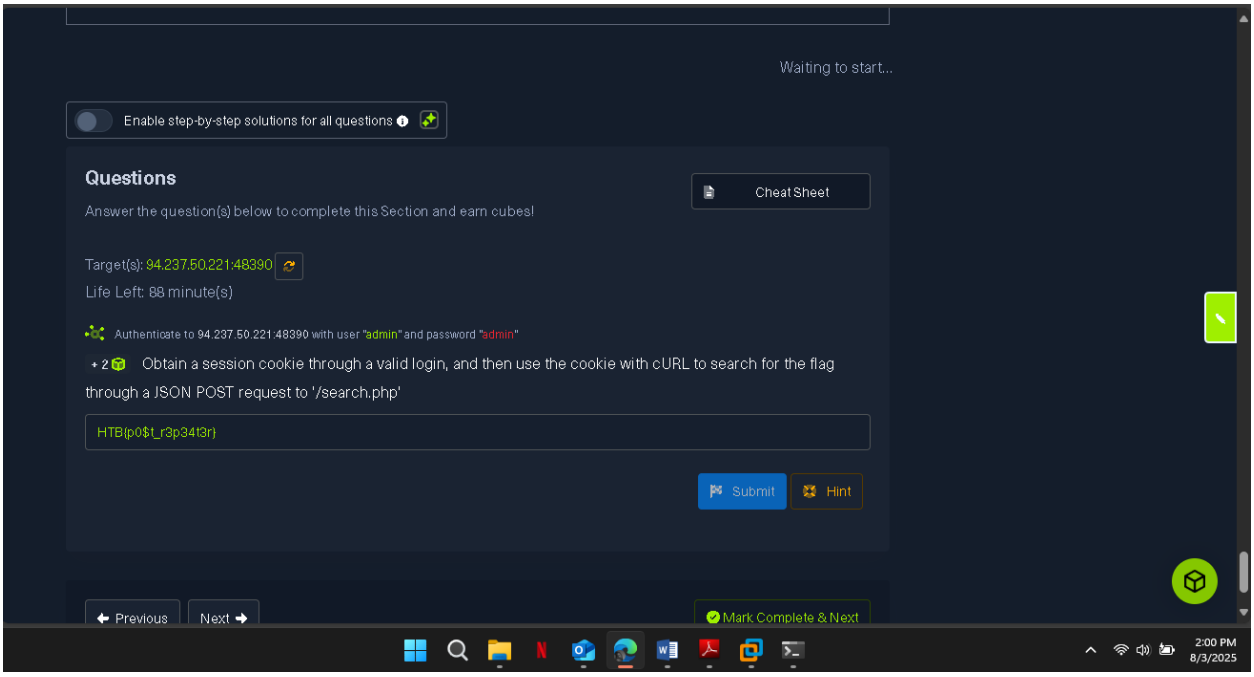

Refresh this page, hold to see more options
flag
Leeds (UK)
Leicester (UK)
Type a city name and hit Enter
Welcome
Elements
Console
Sources
Network
Preserve log
Disable cache
No throttling
Filter
Invert
More filters
All
Fetch/XHR
Doc
CSS
JS
Font
Img
Media
Manifest
Socket
Wasm
Other
1,000 ms
2,000 ms
3,000 ms
4,000 ms
5,000 ms
6,000 ms
7,000 ms
Name
94.237.57.115
reset.min.css
css?family=Roboto:400,500,700
style.css
jquery.min.js
script.js
KFO7CnqEu92Fr1ME7kSn66aG...
search.php?search=flag
Headers
Payload
Preview
Response
Initiator
Timing
General
Request URL
http://94.237.57.115:44057/search.php?search=flag
Request Method
GET
Status Code
200 OK
Remote Address
94.237.57.115:44057
Referrer Policy
strict-origin-when-cross-origin
Response Headers
Raw
HTTP/1.1 200 OK
Date: Sun, 03 Aug 2025 10:48:55 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Length: 15
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
Request Headers
Raw
GET /search.php?search=flag HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
8 requests
1.0 kB transferred
Console
Issues
1:50 PM
8/3/2025

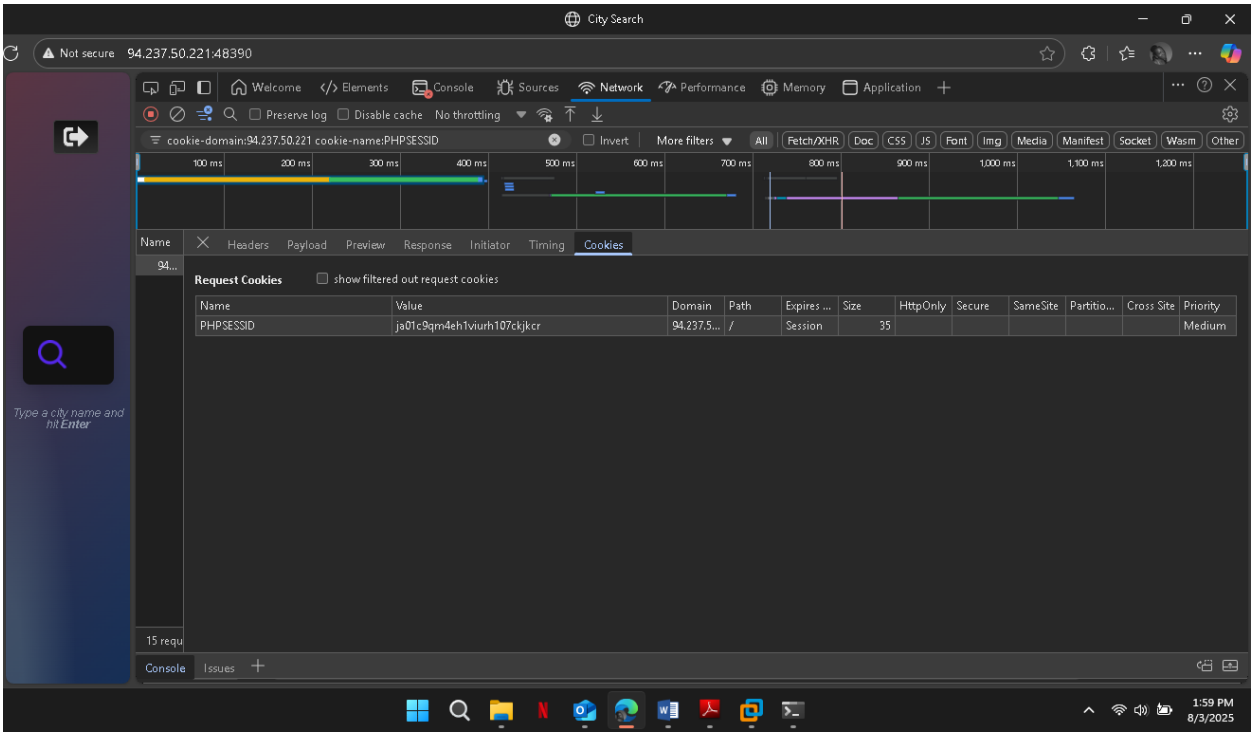

Command Prompt
C:\Users\user>curl -u admin:admin "http://94.237.57.115:44057/search.php?search=flag"
flag: HTB{curl_g3773r}
C:\Users\user>
1:49 PM
8/3/2025

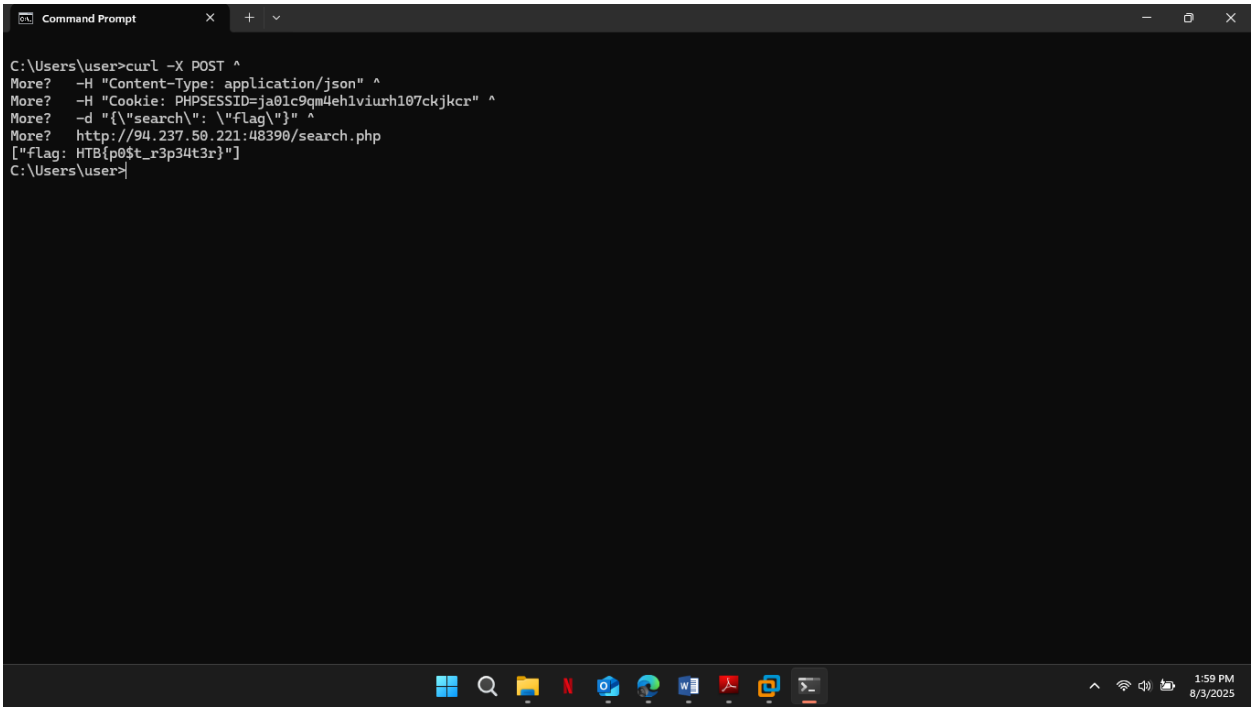


POST

In this section, we examined HTTP POST requests to understand how web applications handle form submissions, authentication, and session management. Unlike GET, POST transmits data in the request body, making it more secure for sending credentials or larger payloads. We simulated a login to a sample web app using cURL, submitting username=admin&password=admin, and observed a successful response indicating access to a search feature. By inspecting browser dev tools and response headers, we identified a session cookie (PHPSESSID) used to maintain authenticated access, which we reused in subsequent cURL and fetch() requests. We also explored interacting with a JSON-based endpoint (search.php) by sending a structured POST request with appropriate headers, successfully retrieving dynamic results like ["London (UK)"]. This exercise highlighted the importance of understanding headers, content types, and cookies in authenticated requests. Ultimately, it demonstrated how tools like cURL, browser devtools, and JavaScript's fetch() enable effective manual testing of web applications and APIs, especially for validating authentication flows and identifying potential vulnerabilities.


Waiting to start...
Enable step-by-step solutions for all questions
Questions
Answer the question(s) below to complete this Section and earn cubes!
Cheat Sheet
Target(s): 94.237.50.221:48390
Life Left: 88 minute(s)
Authenticate to 94.237.50.221:48390 with user "admin" and password "admin"
+ 2 Obtain a session cookie through a valid login, and then use the cookie with cURL to search for the flag through a JSON POST request to '/search.php'
HTB{p0$t_r3p34t3r}
Submit
Hint
Previous
Next
Mark Complete & Next
2:00 PM
8/3/2025

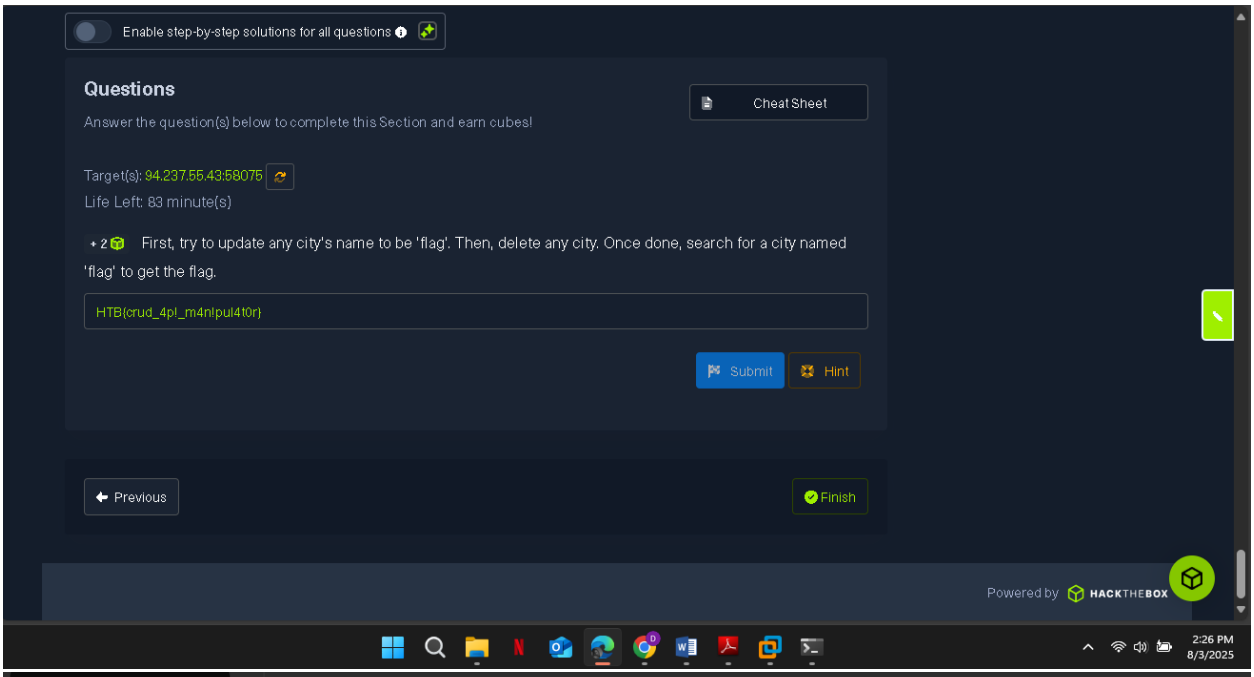

City Search
Not secure 94.237.50.221:48390
Welcome
Elements
Console
Sources
Network
Performance
Memory
Application
Preserve log
Disable cache
No throttling
cookie-domain:94.237.50.221 cookie-name:PHPSESSID
Invert
More filters
All
Fetch/XHR
Doc
CSS
JS
Font
Img
Media
Manifest
Socket
Wasm
Other
Name
Headers
Payload
Preview
Response
Initiator
Timing
Cookies
Request Cookies
show filtered out request cookies
Name
Value
Domain
Path
Expires ...
Size
HttpOnly
Secure
SameSite
Partitio...
Cross Site
Priority
PHPSESSID
ja01c9qm4eh1viurh107ckjkcr
94.237.5...
/
Session
35
Medium
Type a city name and hit Enter
Console
Issues
1:59 PM
8/3/2025

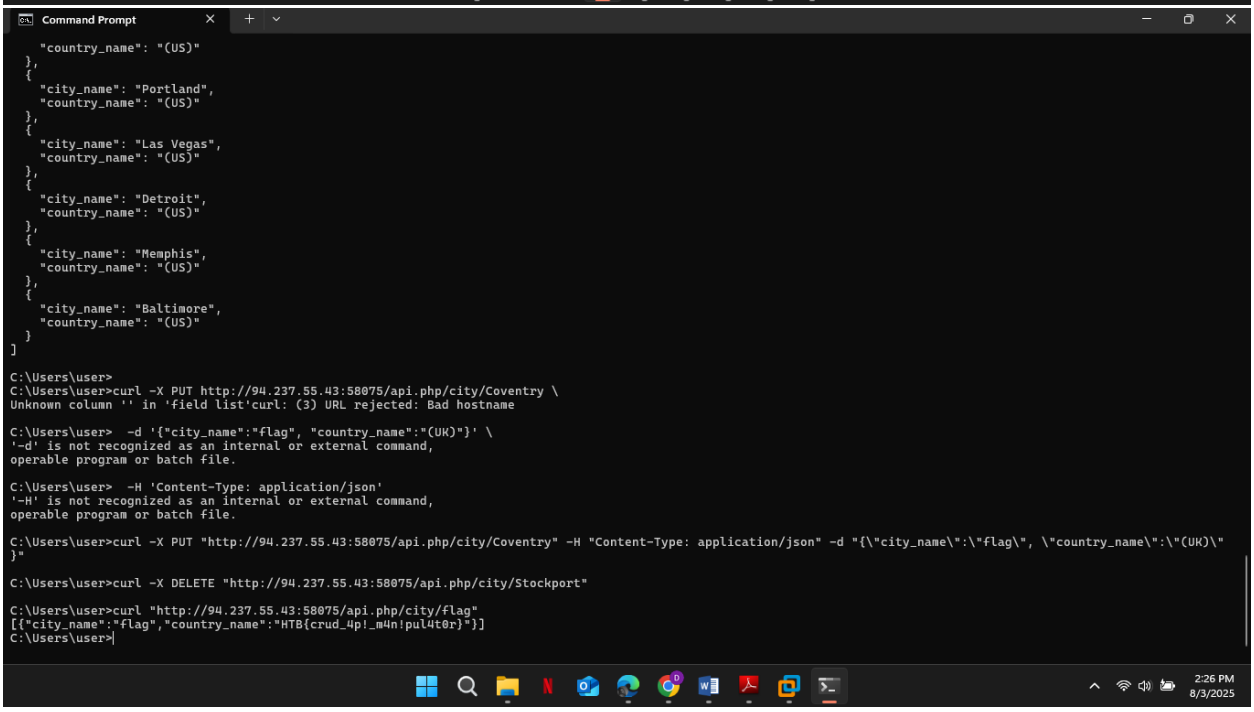


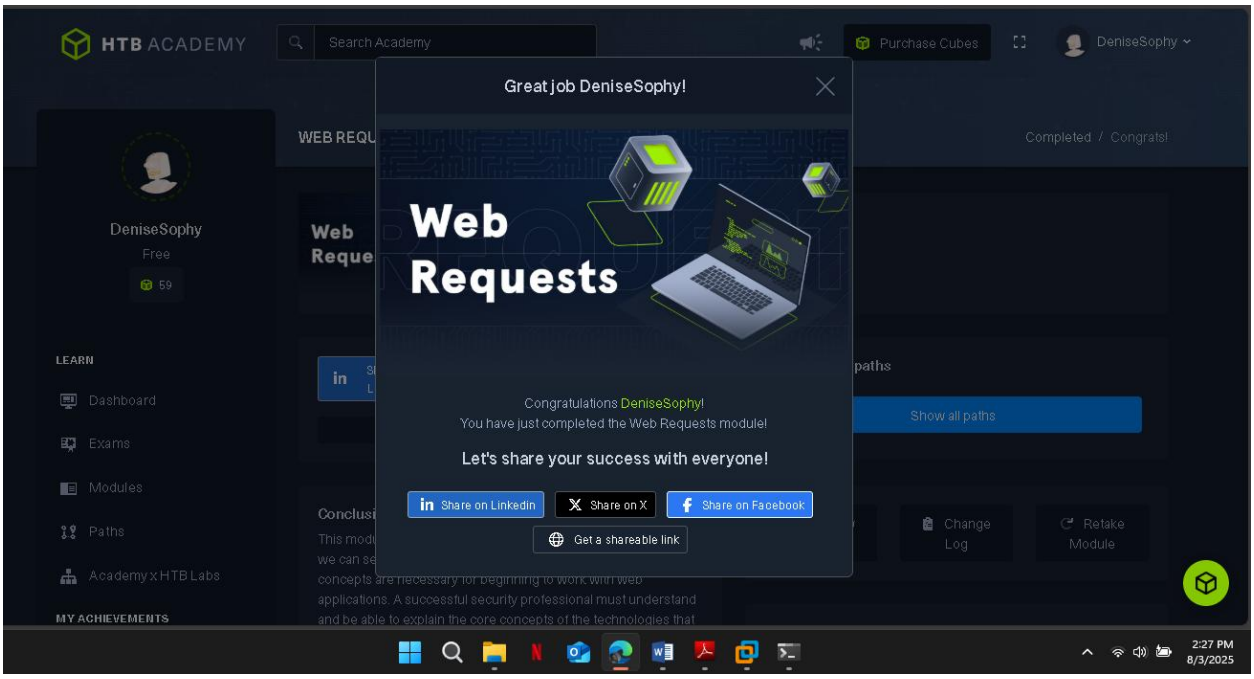
CRUD API

In this section, we explored how modern web applications leverage APIs to perform direct database interactions using standardized HTTP methods in what's commonly referred to as CRUD operations—Create, Read, Update, and Delete. Instead of interacting with the backend through traditional PHP parameters embedded in HTML forms or URL queries, we learned how RESTful APIs streamline this process, allowing us to make database requests programmatically and cleanly via endpoints such as /api.php/city/london. Using tools like curl and jq, we practiced querying specific city data (Read via GET), inserting new city records (Create via POST), modifying existing entries (Update via PUT), and removing records (Delete via DELETE). Each request passed data in JSON format, and proper headers like Content-Type: application/json were used to inform the server about the payload structure. The tutorial also covered essential concepts such as using empty strings to retrieve all entries, leveraging wildcard searches, and observing the result formatting in both raw and pretty-printed JSON output. We also touched on HTTP PATCH vs PUT, where PATCH is for partial updates while PUT is for complete replacements, as well as the importance of proper authentication using cookies or JWTs to enforce access control—highlighting that in real-world APIs, unrestricted access to write, update, or delete endpoints could pose major security risks. Overall, this hands-on exercise demystified

how backend logic is abstracted and secured via APIs while giving us practical insight into how to interface with these endpoints as both developers and ethical hackers.







https://academy.hackthebox.com/achievement/1918539/35

Conclusion

This lab reinforced the foundational role of HTTP and HTTPS in web communication by breaking down and analyzing real-time traffic. By using Wireshark to inspect packet flows and headers, and by leveraging command-line tools like curl for controlled request generation, key elements of the protocol—such as status codes, headers, cookies, and TLS encryption—were demystified. These skills are vital in both offensive and defensive cybersecurity roles, as they allow practitioners to understand how data traverses networks, identify vulnerabilities, and validate secure communication practices. The exercise not only built technical proficiency in traffic analysis but also deepened conceptual knowledge of how the Internet operates beneath the surface.