WEEK3 ASSIGNMENT 2: WEB REQUESTS

RENE OLUOCH

CS-CNS09-25030

https://academy.hackthebox.com/achievement/1914267/35

Table of Contents

Introduction

In this module, I immersed myself in the foundational principles of how web communication works through the HTTP and HTTPS protocols, focusing on how requests and responses are structured, transmitted, and interpreted by both clients and servers. I learned to break down the anatomy of a URL, differentiate between HTTP methods such as GET, POST, PUT, and DELETE, and explore the meaning and importance of HTTP status codes. Using tools like cURL and browser DevTools, I was able to craft, send, and inspect various types of requests, enabling a deeper understanding of how web applications process data and respond to user interactions. I also explored the importance of headers—both for functionality and security—and how authentication mechanisms like HTTP Basic Auth and cookies work in practice. By progressing through real-world exercises, I solidified my understanding of request payloads, JSON formatting, and web session handling. This knowledge culminated in learning how to directly interact with APIs through CRUD operations, illustrating how modern web applications manage data using RESTful principles.

HyperText Transfer Protocol (HTTP)

In this module, I explored how web communication works using HTTP and learned how web clients and servers interact through structured requests and responses. I studied the anatomy of a

URL, understanding its components such as scheme, host, path, query strings, and fragments, and how each plays a role in requesting specific resources from a web server. The process of sending a web request was clearly illustrated—beginning with the browser resolving a domain name via DNS to retrieve the correct IP address, followed by the browser making an HTTP request to the server and receiving a corresponding response with a status code and content, such as HTML. To reinforce this, I used the cURL command-line tool to interact directly with web servers. I practiced sending basic HTTP GET requests, retrieving raw HTML content, and downloading web resources using various cURL options like -O for saving files, -s for silent mode, and -h for help documentation. Unlike web browsers that render pages visually, cURL provides the underlying structure of responses, which is particularly useful in penetration testing where understanding headers and server behavior is crucial. This hands-on experience helped bridge theoretical understanding and practical application, laying a strong foundation for web application analysis and exploitation.

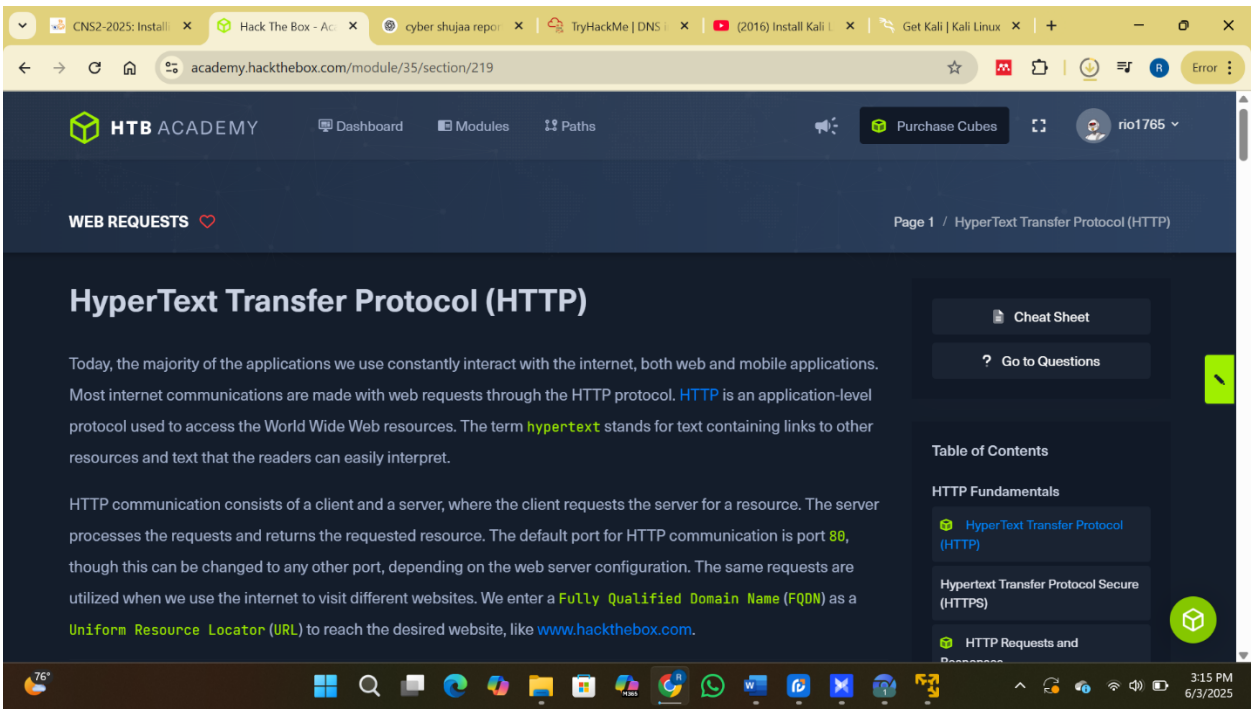

Fig 1.0 http

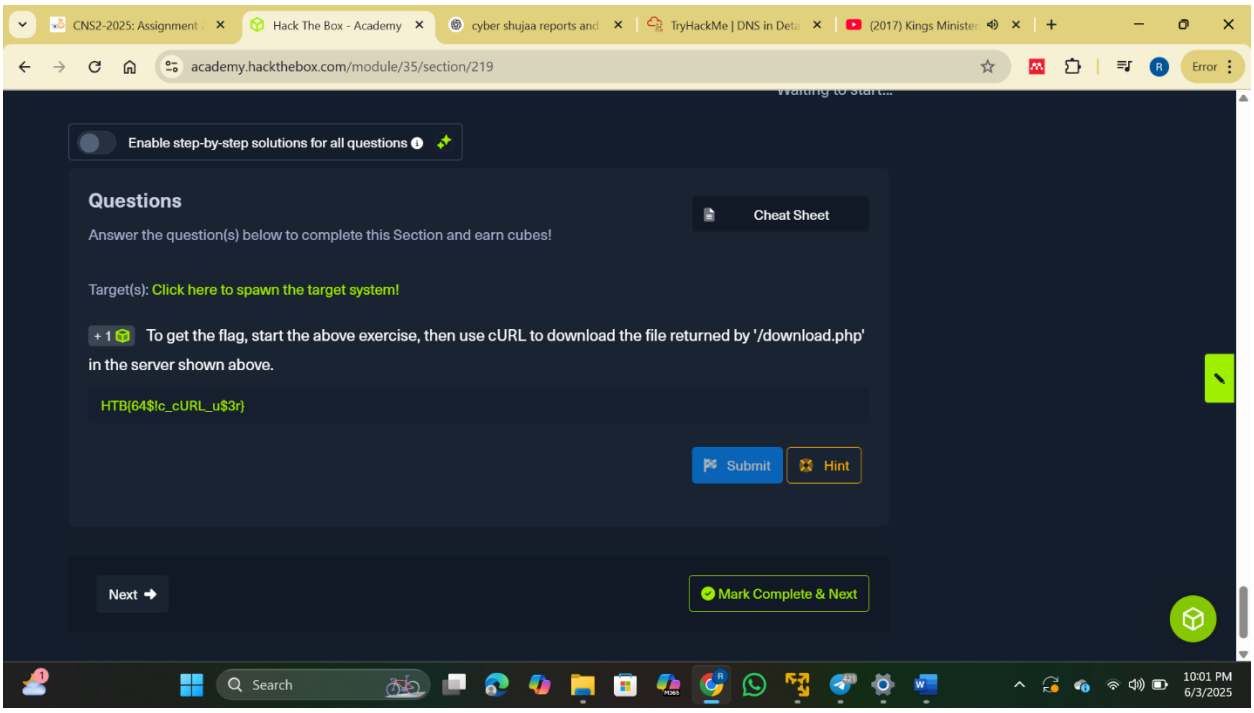

Fig 1.1 question on capturing flag using curl command

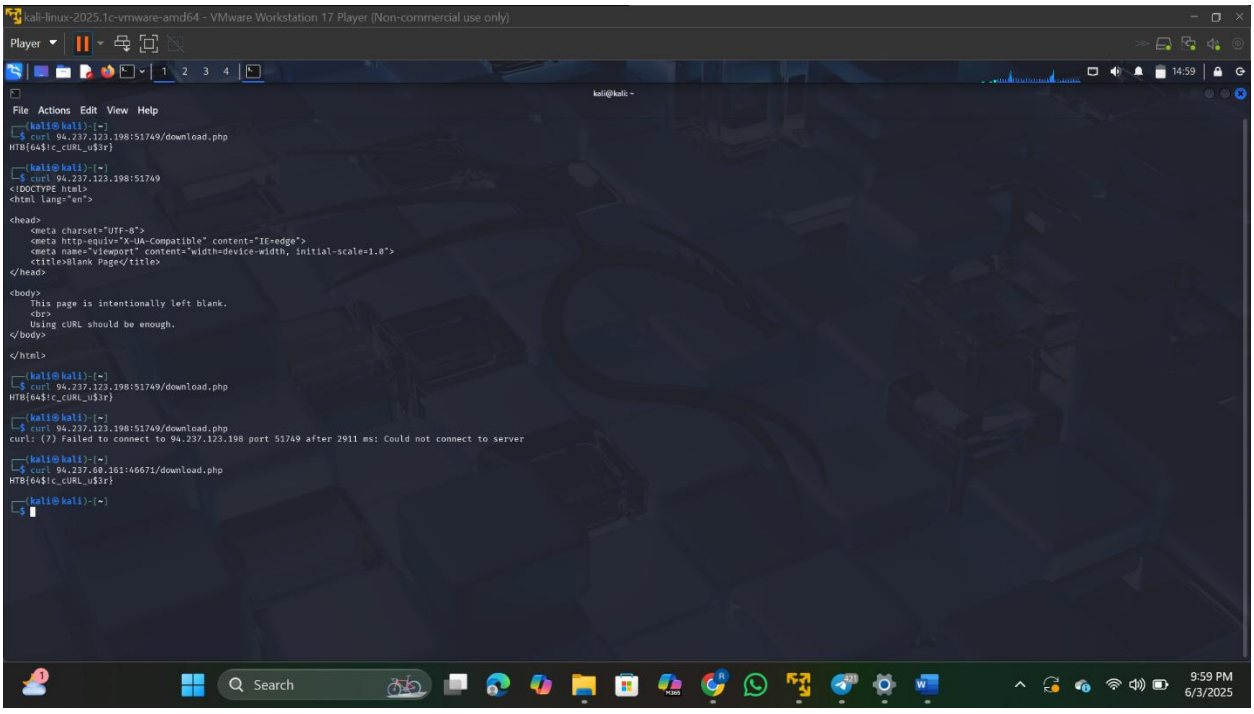

Fig 1.2 capturing flag using curl command

Hypertext Transfer Protocol Secure (HTTPS)

In this module, I built on my understanding of HTTP by exploring HTTPS (HyperText Transfer Protocol Secure), which is designed to address the major security flaw of HTTP—transmitting data in clear text. I learned that HTTPS encrypts all communications between the client and server, protecting sensitive data like login credentials from interception via man-in-the-middle (MITM) attacks. This encryption is especially important when using public networks or accessing secure services. I examined the difference in packet data between HTTP and HTTPS using tools like Wireshark, noting that while HTTP requests expose readable data, HTTPS traffic is encrypted and unintelligible to third parties. The module explained the HTTPS connection process, starting with an HTTP request that's redirected to HTTPS via a 301 status code, followed by a TLS handshake involving client and server hello messages, key exchange, and the establishment of a secure communication channel. I also practiced sending HTTPS requests using cURL. By default, cURL validates SSL certificates to prevent MITM attacks, but I learned how to override this with the -k flag when testing sites with invalid or self-signed certificates, which is common in lab or development environments. This section reinforced the importance of secure communication and gave me practical skills to interact with HTTPS-protected resources confidently and securely.

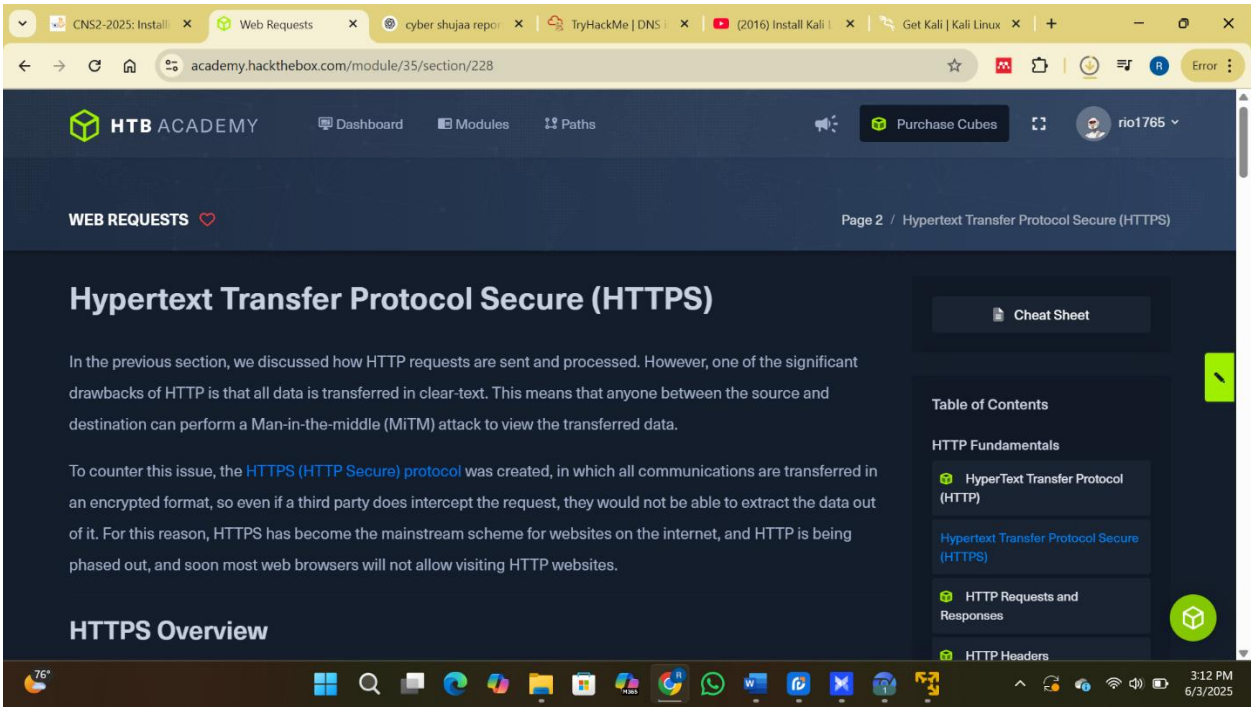

Fig 1.3 https

HTTP Requests and Responses

I explored the structure and flow of HTTP communications, which consist of a request from the client and a response from the server. I examined how an HTTP request includes key elements such as the method (e.g., GET), the path to the resource, the HTTP version, and various headers like Host, User-Agent, and Cookies. These headers can carry critical contextual information that the server uses to process the request. On the server side, the response includes the HTTP version, a status code (such as 200 OK or 401 Unauthorized), response headers, and optionally a body containing content such as HTML, JSON, or other media. I practiced using cURL with the -v and -vvv flags to view full HTTP request and response details, which is particularly useful during web penetration testing for understanding how a server responds under different conditions. Additionally, I used browser DevTools—specifically the Network tab—to monitor live web requests and analyze status codes, headers, and raw response bodies. This allowed me to dissect browser behavior and gain hands-on experience with HTTP mechanics in real-world scenarios. Through this, I developed a solid understanding of how web communication functions and how to investigate it using both command-line tools and graphical interfaces, as I sent a GET request to a server which I was given, read the response header to find version of the apache running on the sevrev

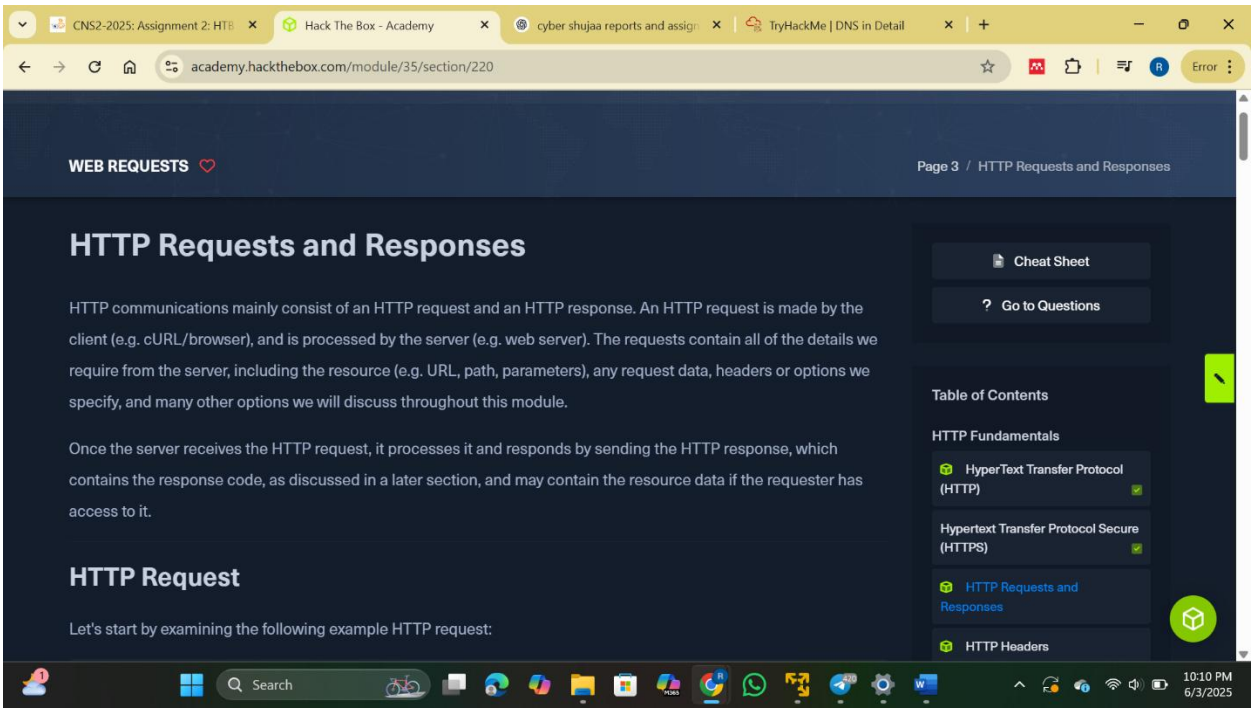

Fig 1.4 http request and response

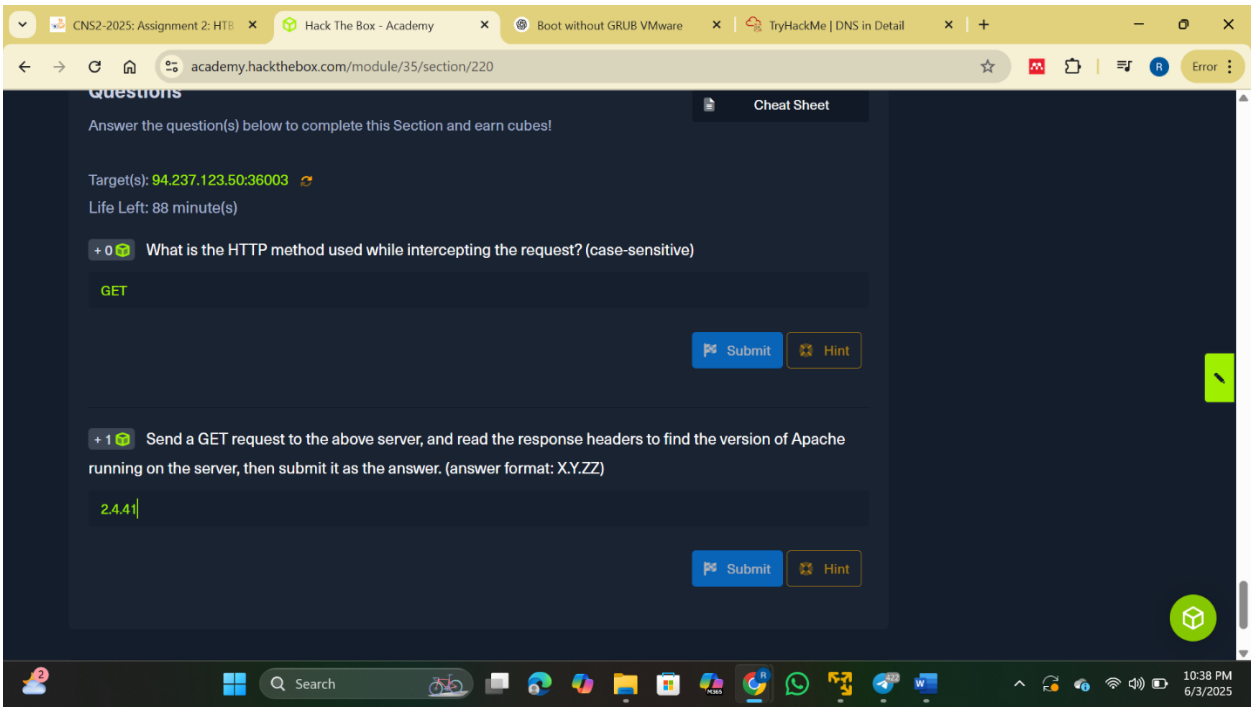

Fig 1.5 questions on request and response

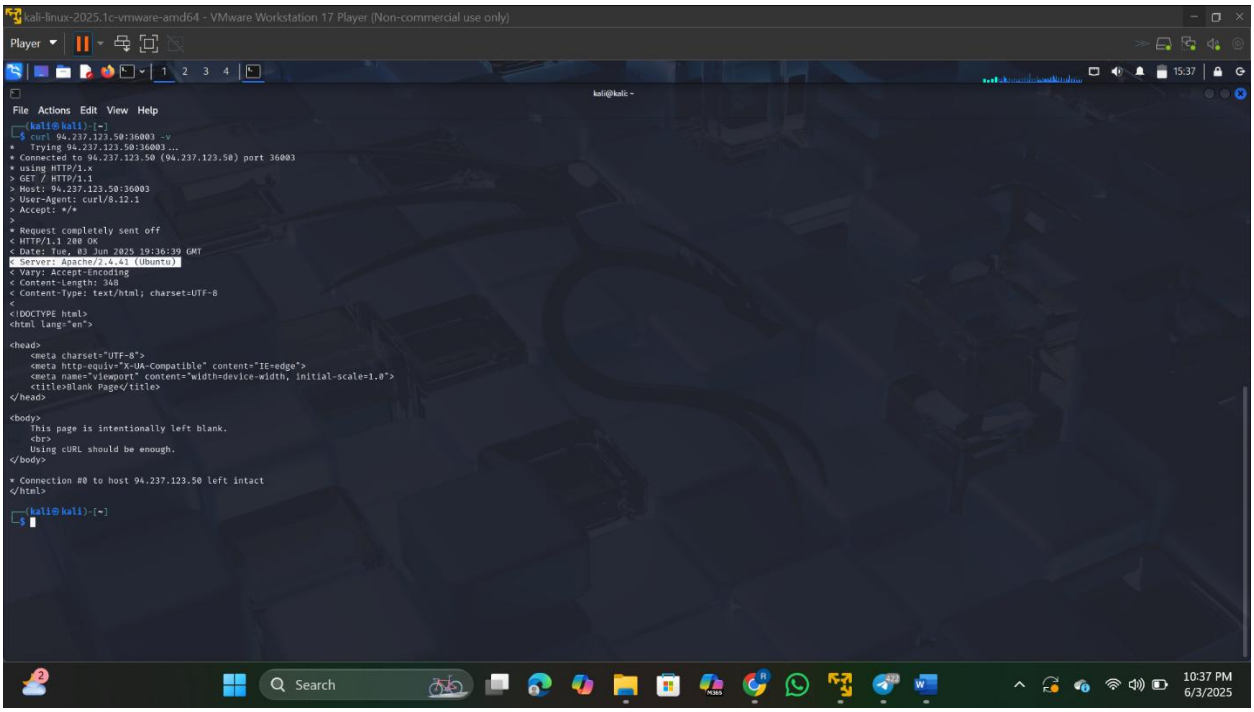

Fig 1.6 finding the version of Apache running the server

HTTP Headers

In this section of the module, I delved into the structure and roles of HTTP headers in client-server communication. I learned that headers are crucial pieces of metadata that guide the handling of HTTP requests and responses. These headers are categorized into five types: general headers, which apply to both requests and responses; entity headers, which describe the content being sent; request headers, which define client-specific metadata such as User-Agent, Host, and Authorization; response headers, which provide context about the server and content such as Set-Cookie, Server, and WWW-Authenticate; and security headers, like Content-Security-Policy and Strict-Transport-Security, which are essential for protecting users against attacks such as XSS and man-in-the-middle threats. Using cURL, I practiced viewing and manipulating headers with options like -I to view response headers only, -i for combined headers and body, and -A to change the user-agent string. This hands-on approach helped me understand how requests are shaped and how servers respond based on the header content. Additionally, I used the browser's DevTools Network tab to visually inspect both request and response headers. The headers were organized into intuitive sections, and I could also view them in raw format. This experience emphasized how essential headers are for authentication, content negotiation, session management, and security in HTTP-based communication.

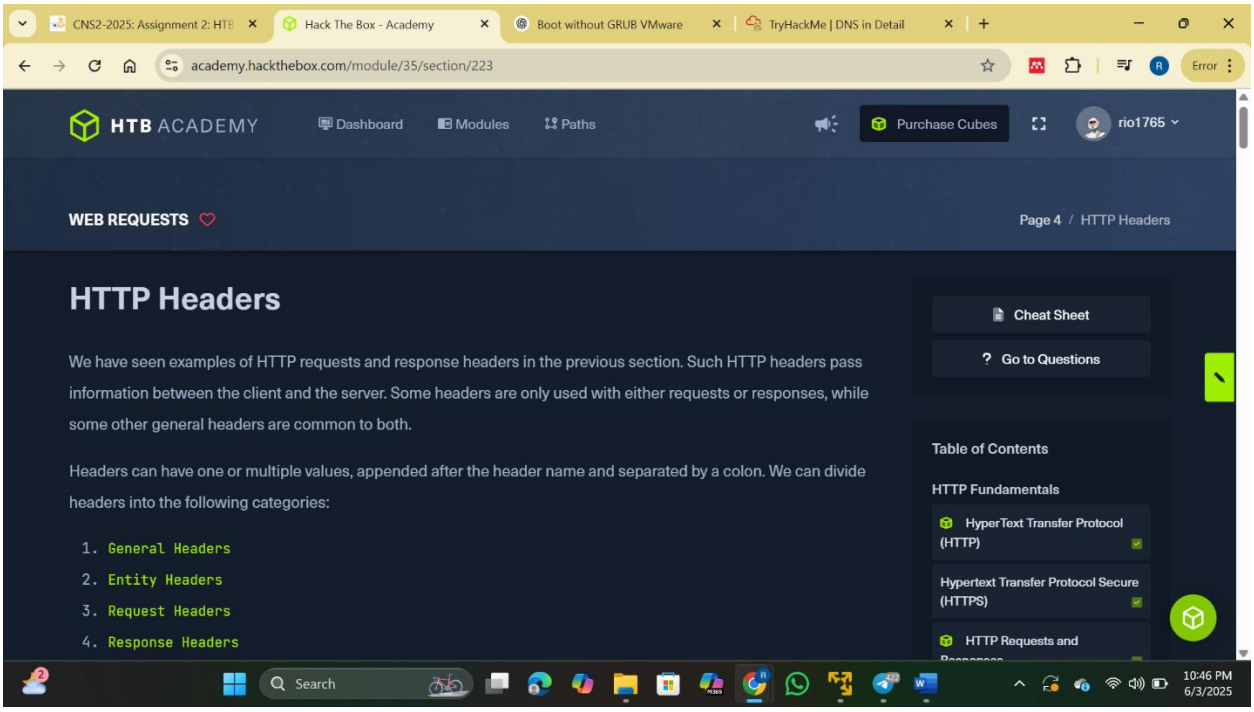

Fig 1.7 http headers

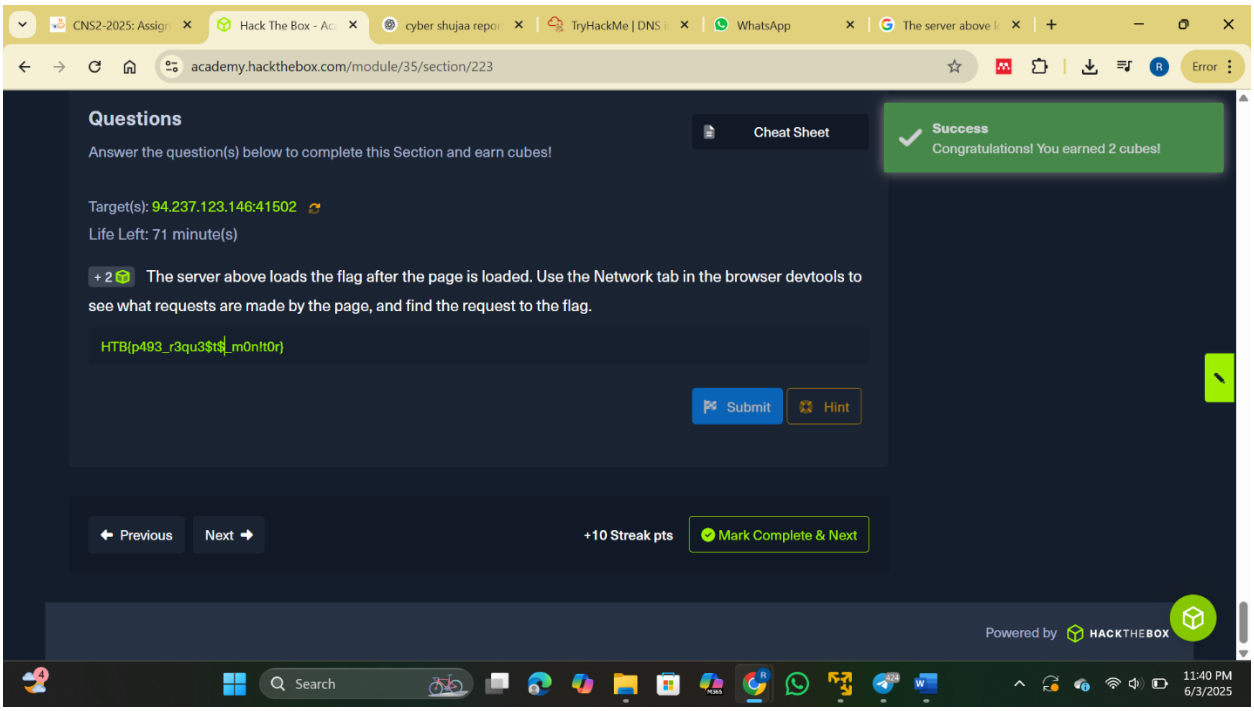

Fig 1.8 questions to http headers

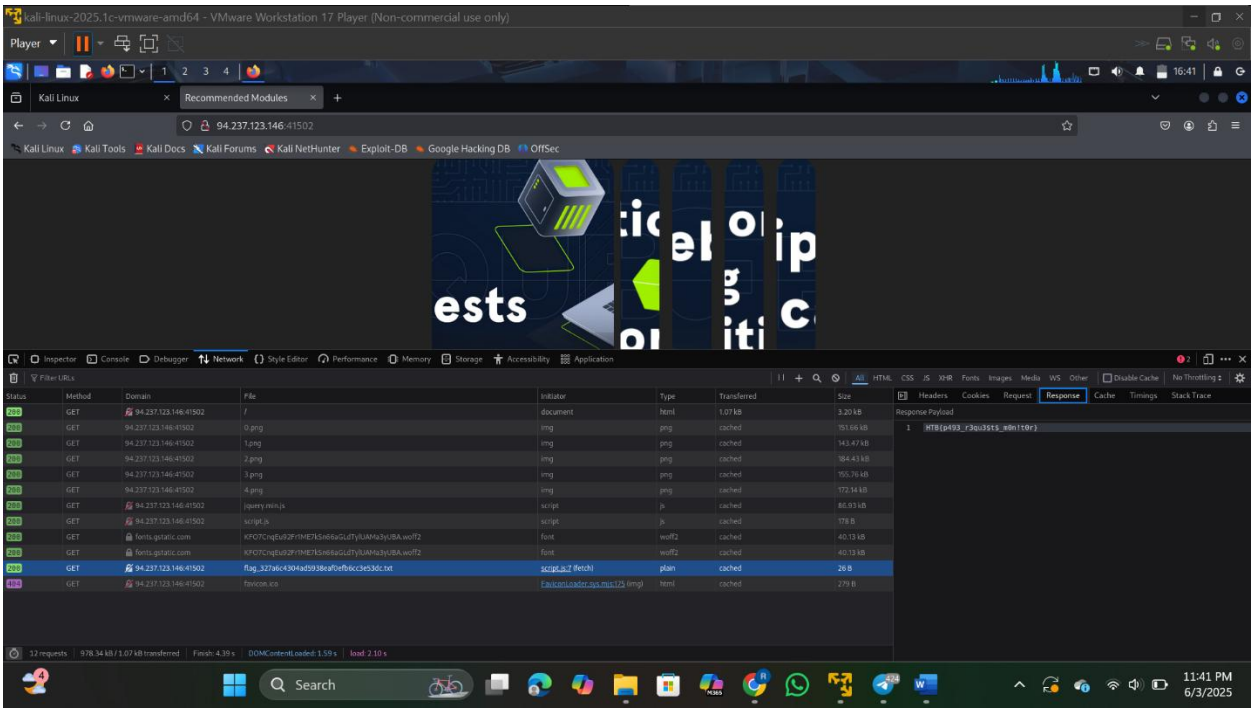

Fig 1.9 finding requests to the flag using dev tools

HTTP Methods and Codes

I explored the essential concepts behind HTTP request methods and response codes, which play a critical role in how web clients and servers communicate. I learned that HTTP methods, like GET, POST, PUT, and DELETE, define the type of action a client wants to perform on a server resource. For example, GET retrieves data, POST sends data in the request body—typically for form submissions or file uploads—while PUT and DELETE are more powerful methods used to modify or remove server-side data, particularly in RESTful APIs. Additionally, methods like HEAD and OPTIONS are used for diagnostics or retrieving metadata about the resource. Using tools like cURL and browser DevTools, I was able to view these methods in action by inspecting the first line of HTTP requests and observing how servers respond. Alongside methods, I studied HTTP response codes, which inform the client of the result of their request. These codes are categorized into five groups: 1xx for informational responses, 2xx for successful interactions like 200 OK, 3xx for redirections such as 302 Found, 4xx for client-side errors like 403 Forbidden or 404 Not Found, and 5xx for server-side issues like 500 Internal Server Error. Understanding these response codes helps in diagnosing issues with requests or misconfigured resources. Overall, this section sharpened my ability to read, craft, and interpret HTTP requests and responses—skills that are foundational for effective web penetration testing and troubleshooting.

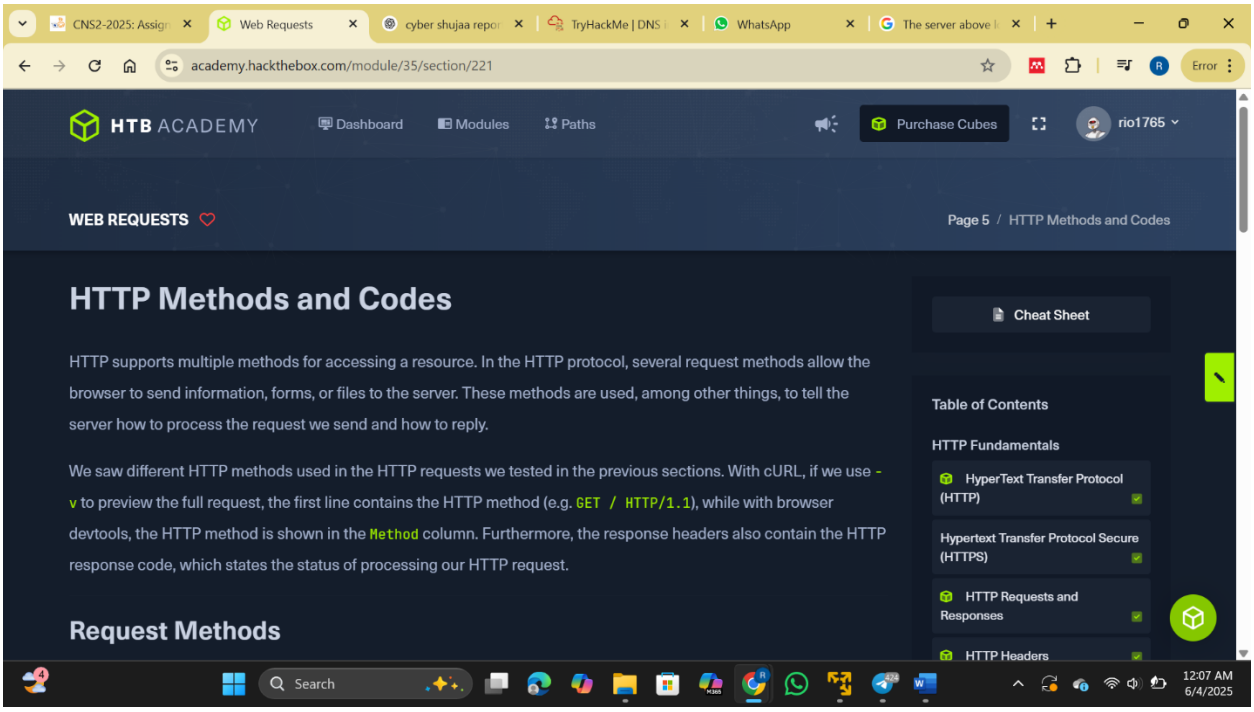

Fig 1.10 http methods and codes

GET

Exploring the practical application of HTTP methods and authentication mechanisms using both browser DevTools and the cURL command-line tool, I began by observing that browsers

typically initiate a GET request when visiting a URL, and then follow up with additional HTTP requests to load page assets or make dynamic queries. To understand these interactions, I used the Network tab in DevTools, which showed the methods used, request URLs, and their corresponding responses.

A key focus in this section was HTTP Basic Authentication, which differs from traditional login forms by embedding credentials directly into the HTTP request headers. When I visited a protected endpoint in the exercise, I was prompted to authenticate with a username and password (admin:admin). Initially, an unauthenticated GET request returned a 401 Unauthorized response along with a WWW-Authenticate header indicating the need for Basic authentication. I then used cURL's -u flag to supply credentials and gain access. Alternatively, I authenticated successfully by embedding the credentials directly into the URL format (http://admin:admin@<IP>:<PORT>/) or by setting the Authorization header manually using the -H flag with a base64-encoded string of admin:admin.

Once authenticated, I observed how GET parameters were used in dynamic search functions. By interacting with a city search feature on the webpage and monitoring network activity, I saw requests being made to search.php with a query string like ?search=le. Using the "Copy as cURL" feature in DevTools, I replicated the exact request in my terminal, adjusting the search term to flag to uncover hidden data. This hands-on approach demonstrated how requests can be inspected, replicated, and modified to retrieve data directly from backend endpoints. It also emphasized the usefulness of copying requests as Fetch commands to test them within the browser's JavaScript console.

Through these exercises, I gained practical knowledge of how HTTP requests are structured, how authentication headers are applied, and how to use developer tools and command-line utilities to uncover and manipulate HTTP communication in real-world web applications.

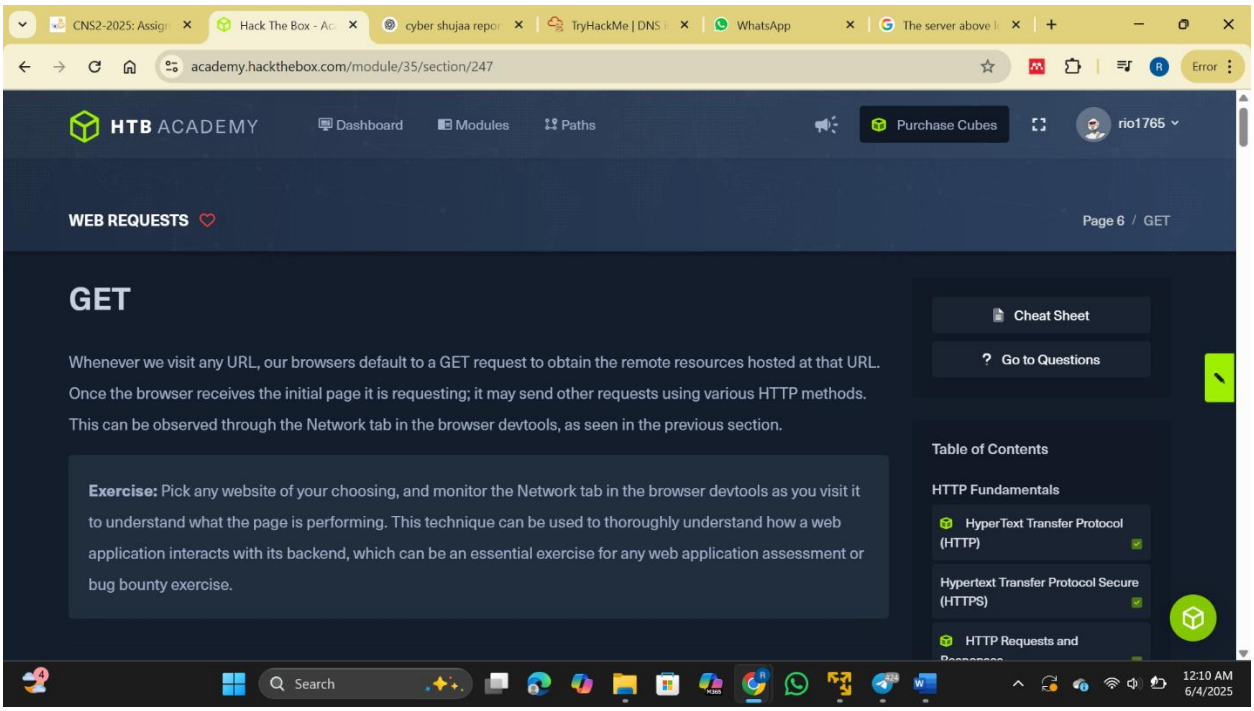

Fig 1.11 get

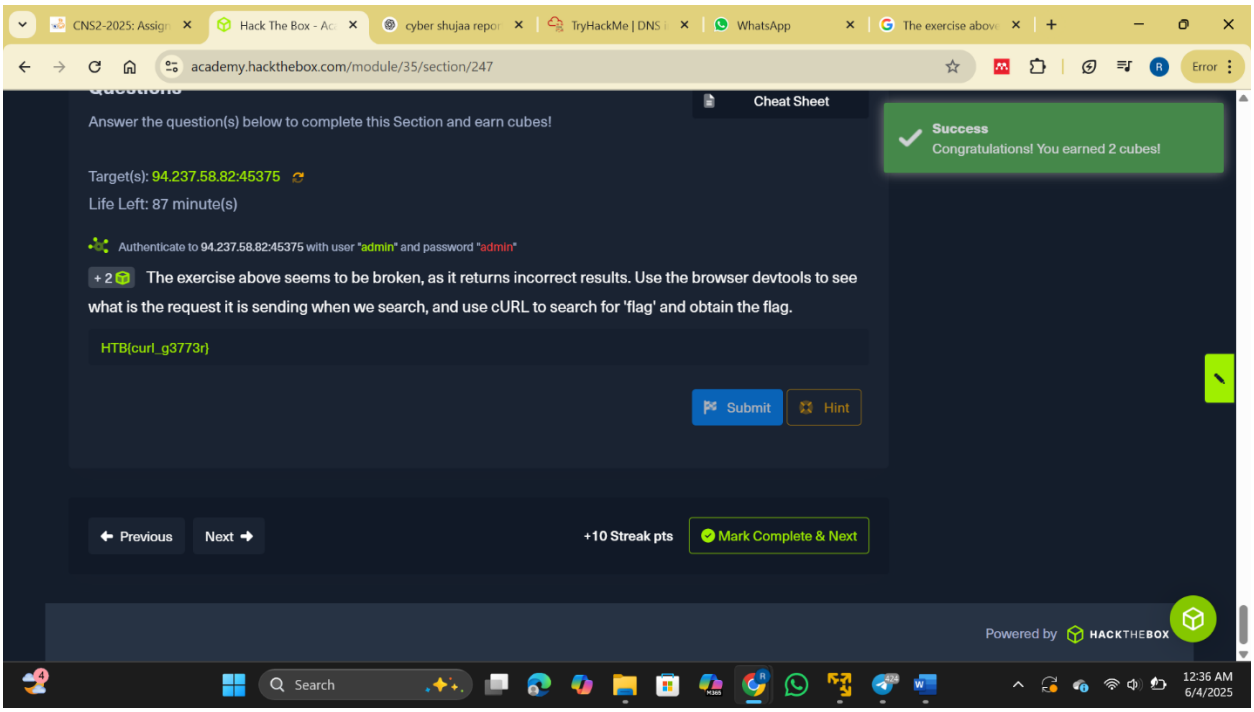

Fig 1.12 questions on get

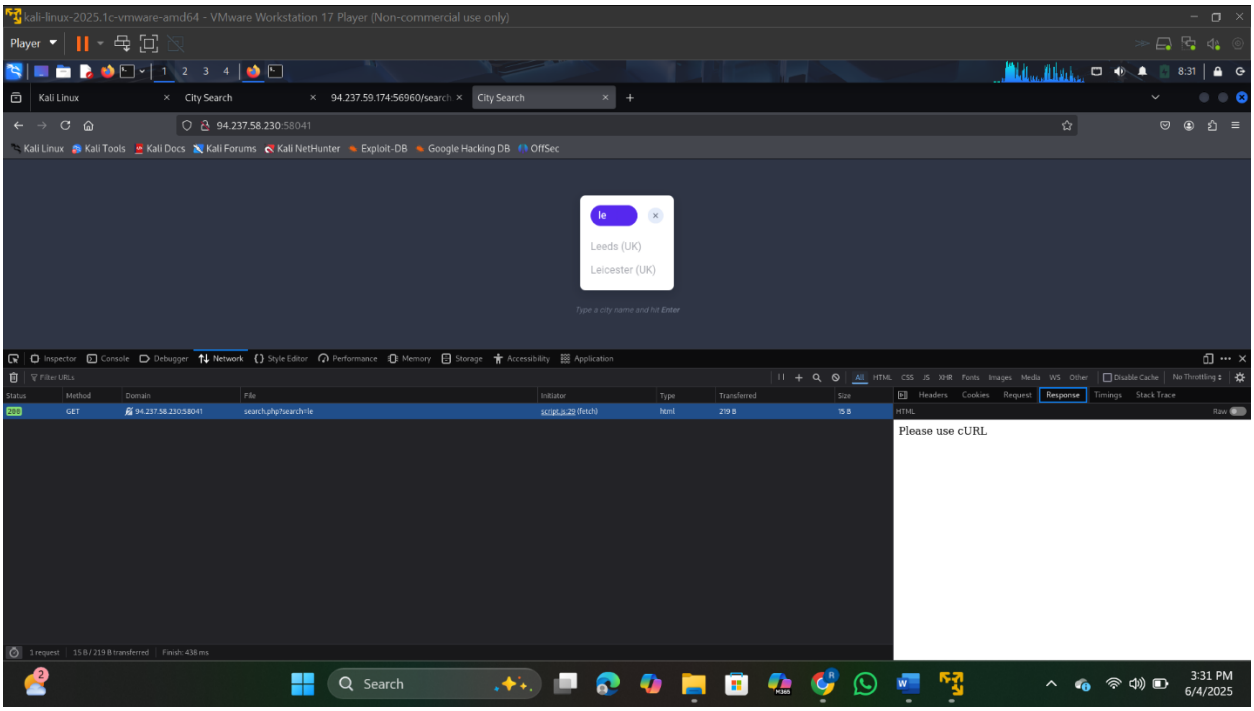

Fig 1.13 request sent when we search

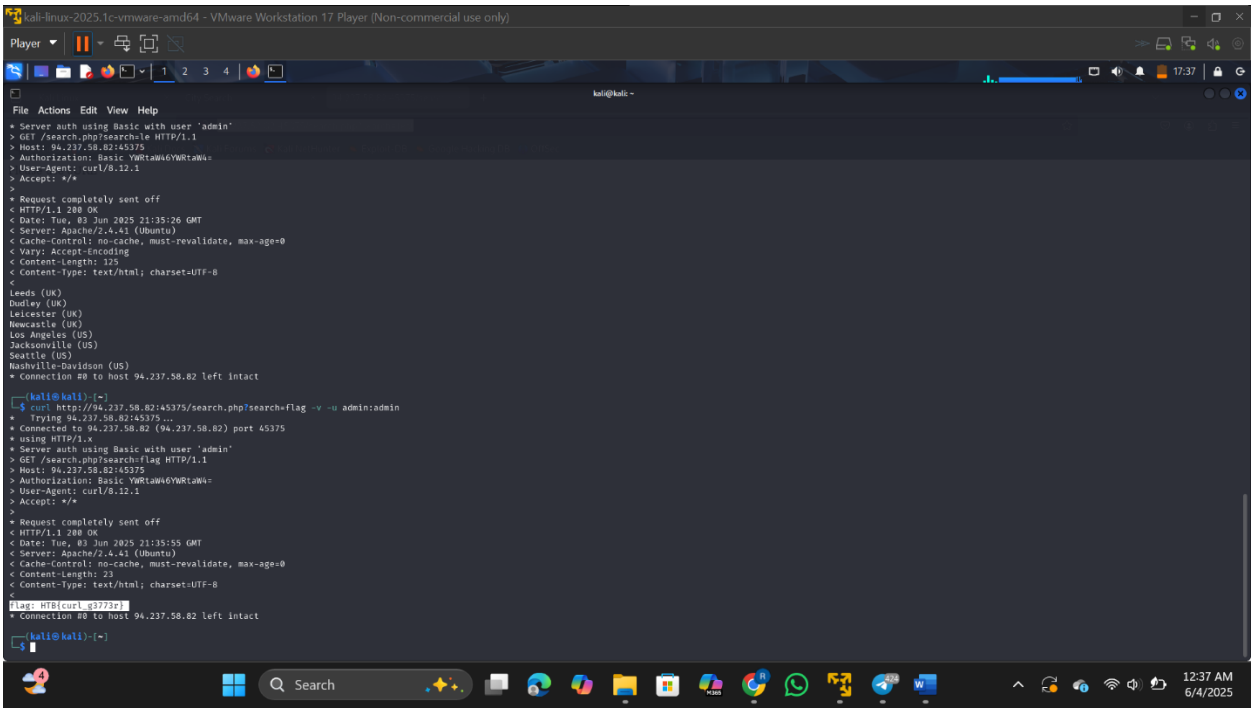

Fig 1.14 using the sent request with curl to search for flag

POST

Here, I explored both theoretical and hands-on aspects of the HTTP protocol, a foundational concept in web application security and penetration testing. The journey began with an in-depth look at the structure of HTTP and HTTPS, dissecting their flow, anatomy, and the way browsers interact with servers through request and response cycles. I learned how components of a URL—including scheme, host, path, query, and fragments—contribute to the way a browser communicates with web servers. I saw firsthand how DNS resolution is an integral first step in this process, retrieving the IP address of the target server before an HTTP request is made.

Working with cURL, a powerful command-line tool for web requests, I was able to simulate HTTP interactions, send GET and POST requests, download content, include custom headers, and observe verbose output to monitor the behavior of both clients and servers. The -v, -i, and -I flags allowed me to capture and analyze full requests, headers, and responses. Meanwhile, using browser DevTools gave me a more visual and interactive perspective, helping me inspect requests in real-time, examine cookies, and replicate HTTP interactions through features like "Copy as cURL" and "Copy as Fetch."

As I advanced, I explored HTTP headers in detail—categorizing them into general, entity, request, response, and security headers. This knowledge proved valuable when I was able to recognize authentication methods (like Basic Auth), track cookies (Set-Cookie, Cookie), and understand protections such as Content-Security-Policy and Strict-Transport-Security. Using cURL's -H and -A flags, I practiced setting and modifying headers like User-Agent and Authorization, which provided hands-on insights into how browsers and APIs communicate securely and contextually.

When examining HTTP methods, I learned how each method—from GET and POST to PUT, DELETE, and OPTIONS—serves a different function in web applications, especially in RESTful APIs. I observed how GET is used to retrieve data, while POST enables users to send form data or JSON payloads. Using exercises in a browser-based lab environment, I authenticated to applications using Basic Auth, extracted session cookies, and reused them in further requests to maintain authenticated sessions.

The culmination of this module was in practicing advanced POST requests using JSON. After successfully logging in using a POST request with credentials passed in the request body, I captured the returned session cookie and used it to send a JSON-encoded POST request to a search endpoint (search.php). This task required setting the Content-Type: application/json header and the Cookie header, allowing me to simulate a dynamic search without using the web interface. Using both cURL and the browser's Fetch API, I retrieved results directly—demonstrating how backend logic can be manipulated independently of frontend restrictions.

Altogether, this module offered a comprehensive and practical foundation in web request analysis. I walked away with the ability to inspect, construct, and manipulate HTTP traffic for

purposes ranging from troubleshooting and automation to security testing and vulnerability discovery.

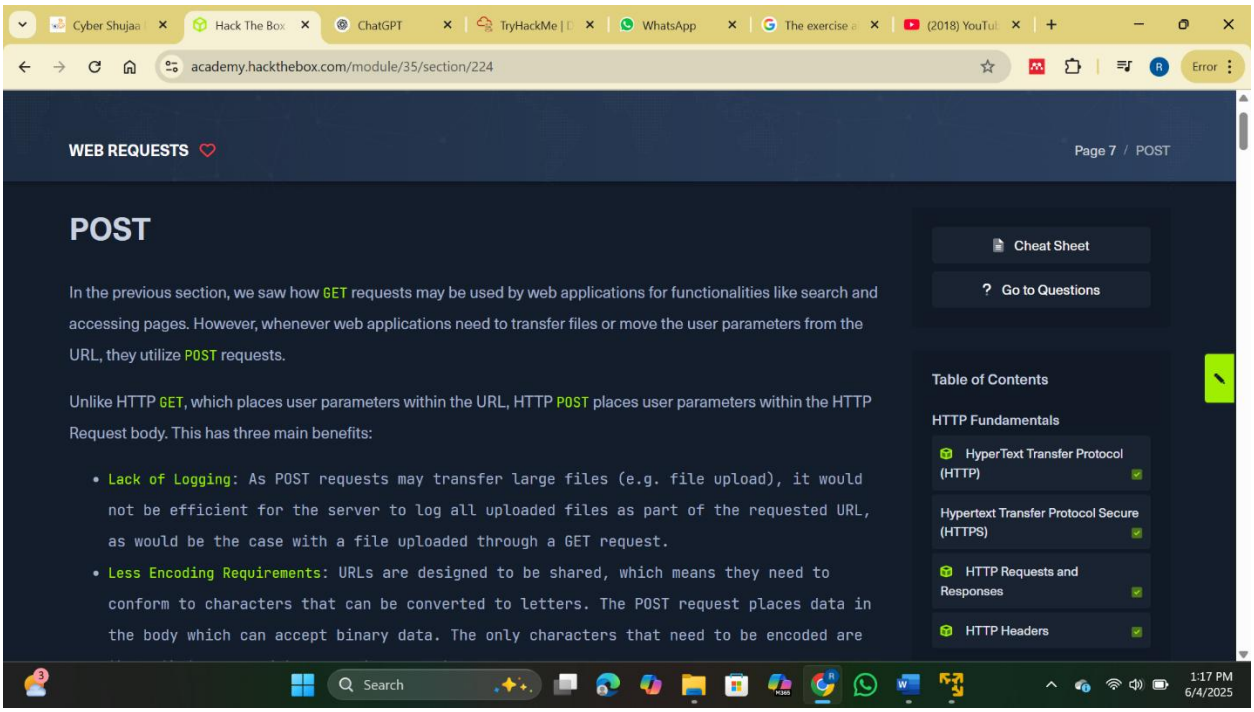

Fig 1.15 post

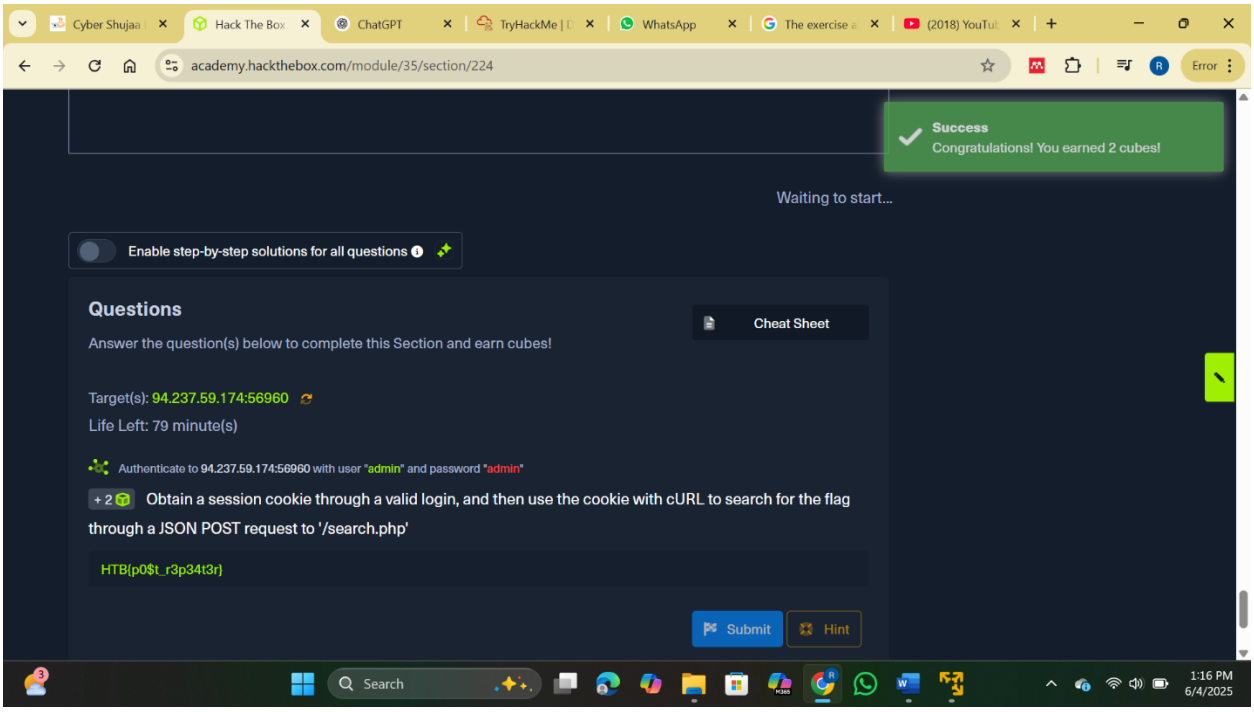

Fig 1.16 question on using curl to search for flag

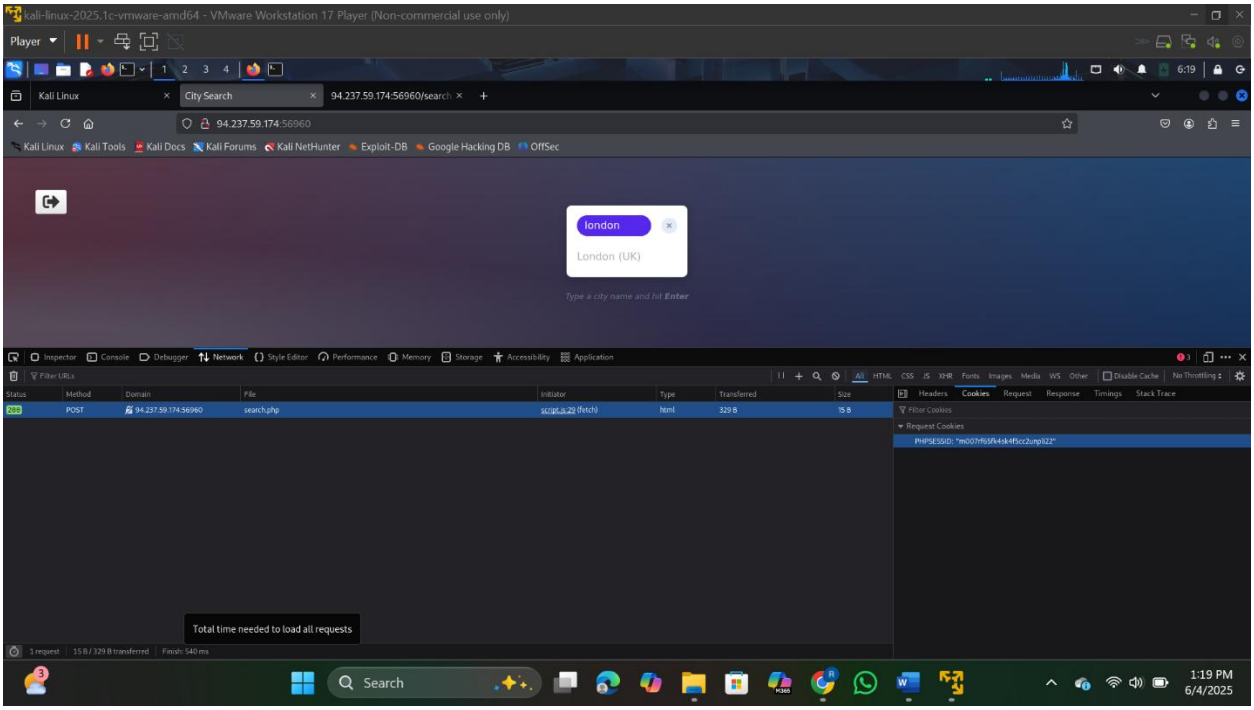

Fig 1.17 obtaining cookie through valid login

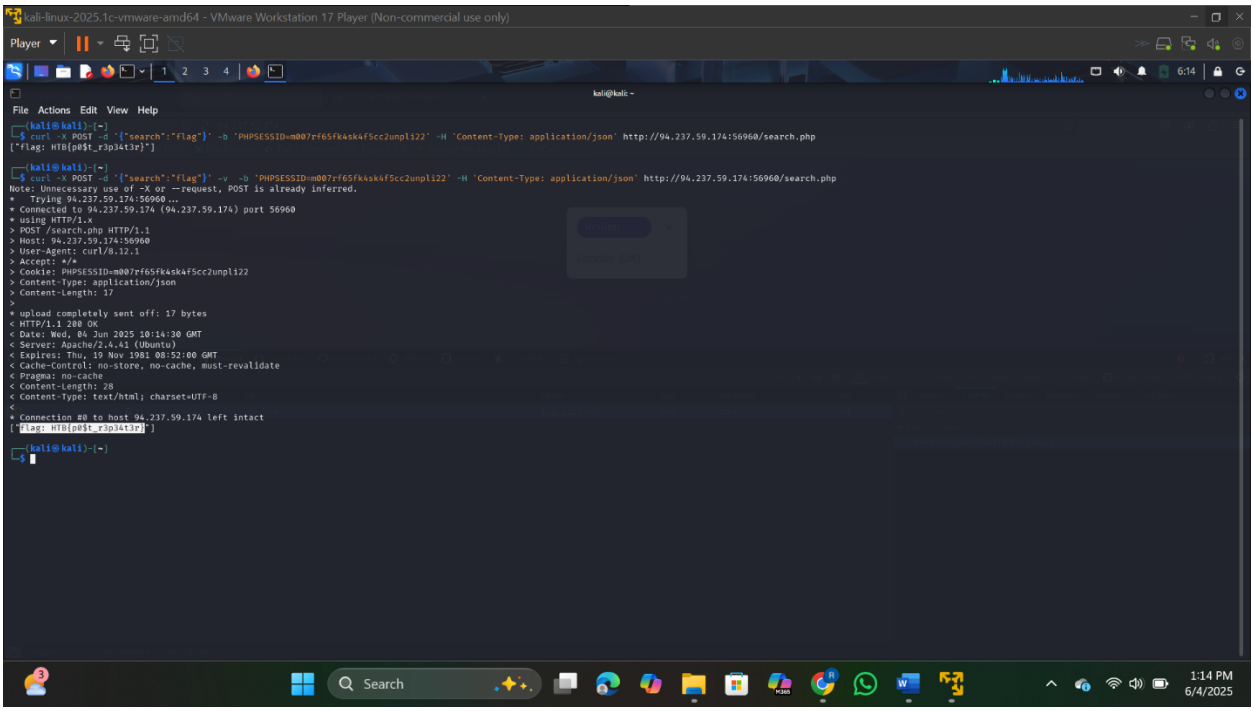

Fig 1.18 using curl to search for flag through a valid cookie

CRUD API

In this section of the module, I delved into the practical applications of CRUD operations (Create, Read, Update, Delete) through APIs using HTTP methods such as GET, POST, PUT, and DELETE. These operations allowed me to interact directly with a backend database via a web API endpoint (api.php). To retrieve (read) data, I used the GET method and queried specific entries like /city/london or used wildcards and blank searches to list broader sets of cities. I learned to format the JSON response using the jq utility(which I hqd to download in kali linux) for clarity. To create a new entry, I used a POST request, setting the Content-Type header to application/json and sending city data in the request body, which successfully added a new city to the table. I confirmed this addition by querying it afterward. Updating a record was achieved with a PUT request targeting a specific city, and the server replaced the old entry with the new data. Finally, deletion was done with a DELETE request, after which a follow-up GET query showed an empty response, confirming the record's removal. These tasks helped reinforce how RESTful APIs operate and how cURL can be used for full database interaction from the command line. The exercise also highlighted how APIs can expose vulnerabilities if authentication and authorization controls are not enforced properly.

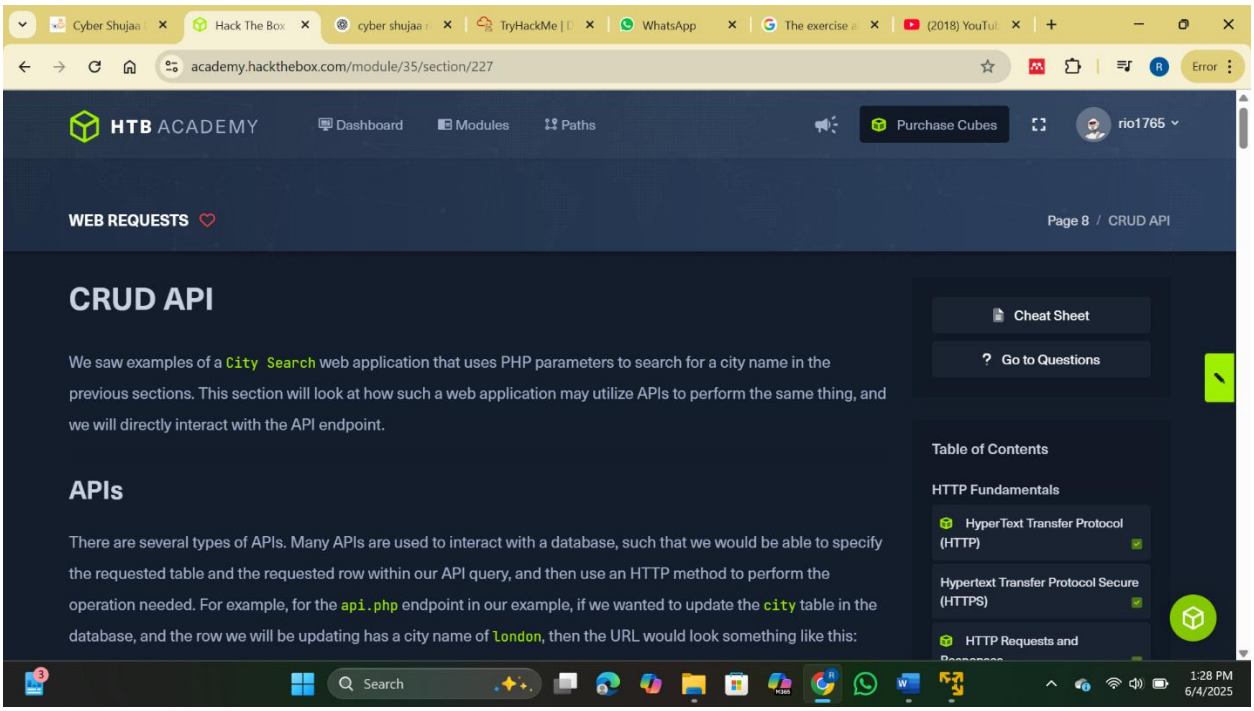

Fig 1.19 crud api

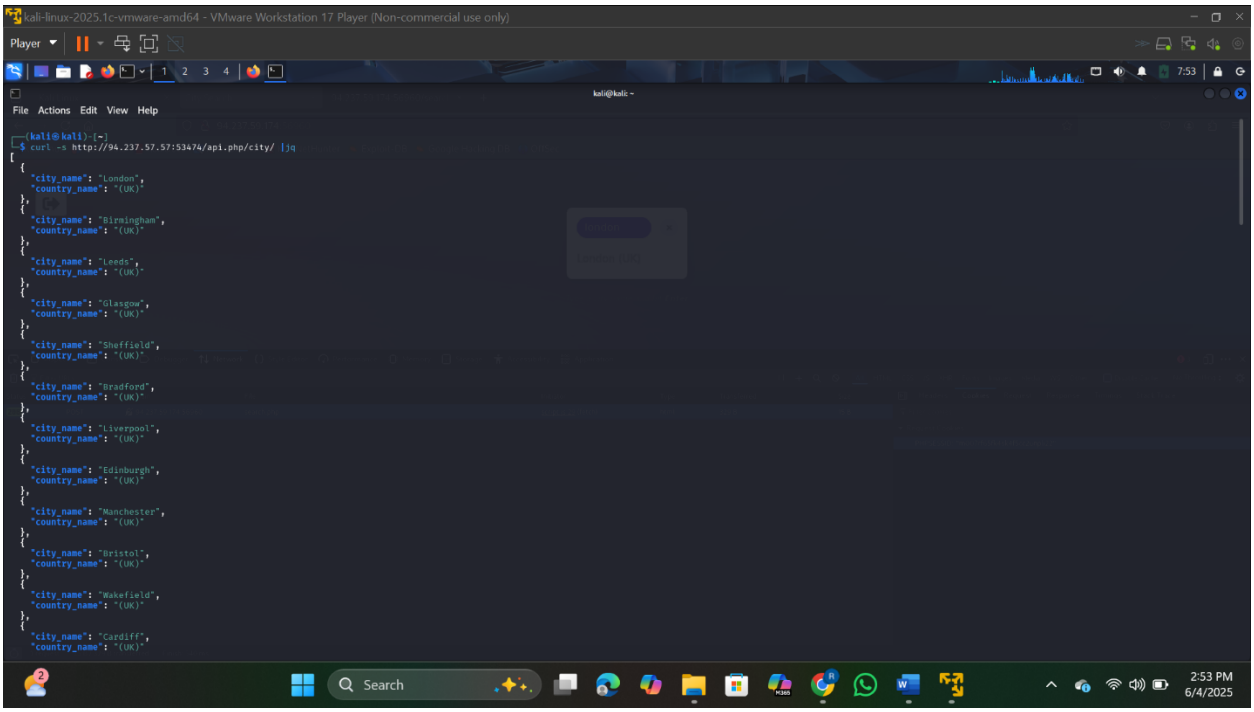

Fig 1.20 viewing all cities after passing empty string

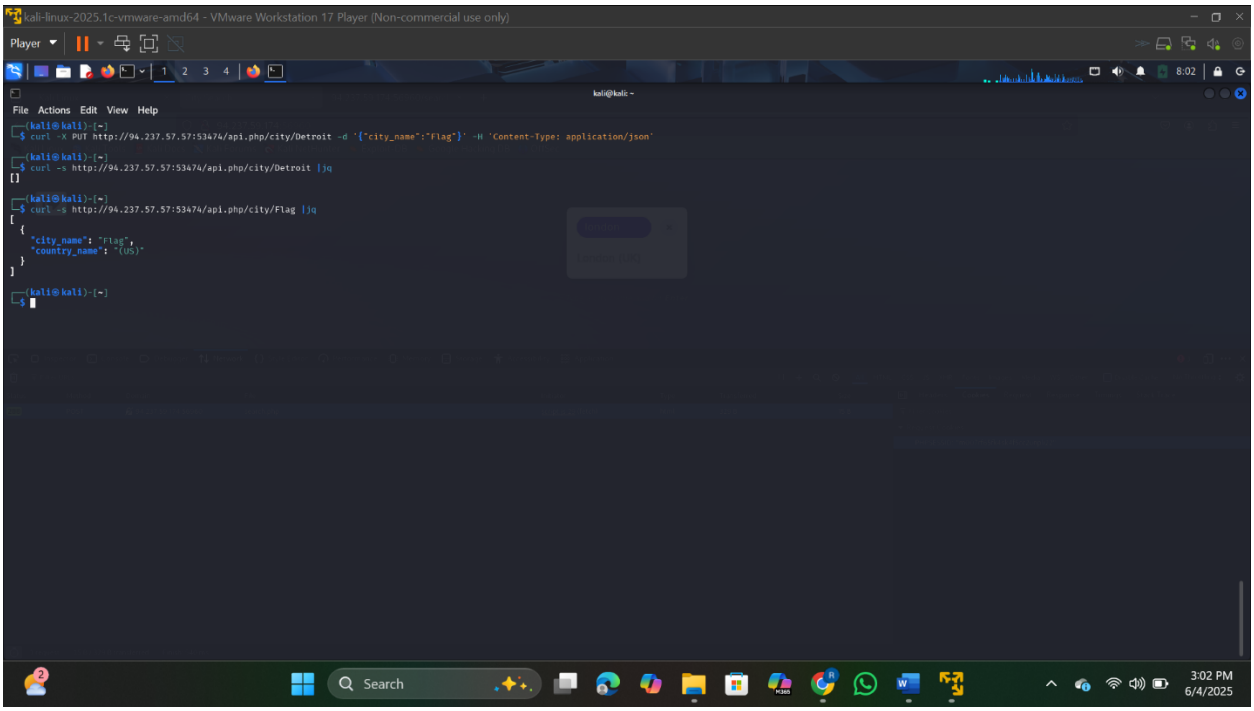

Fig 1.21 renaming city Detroit to flag

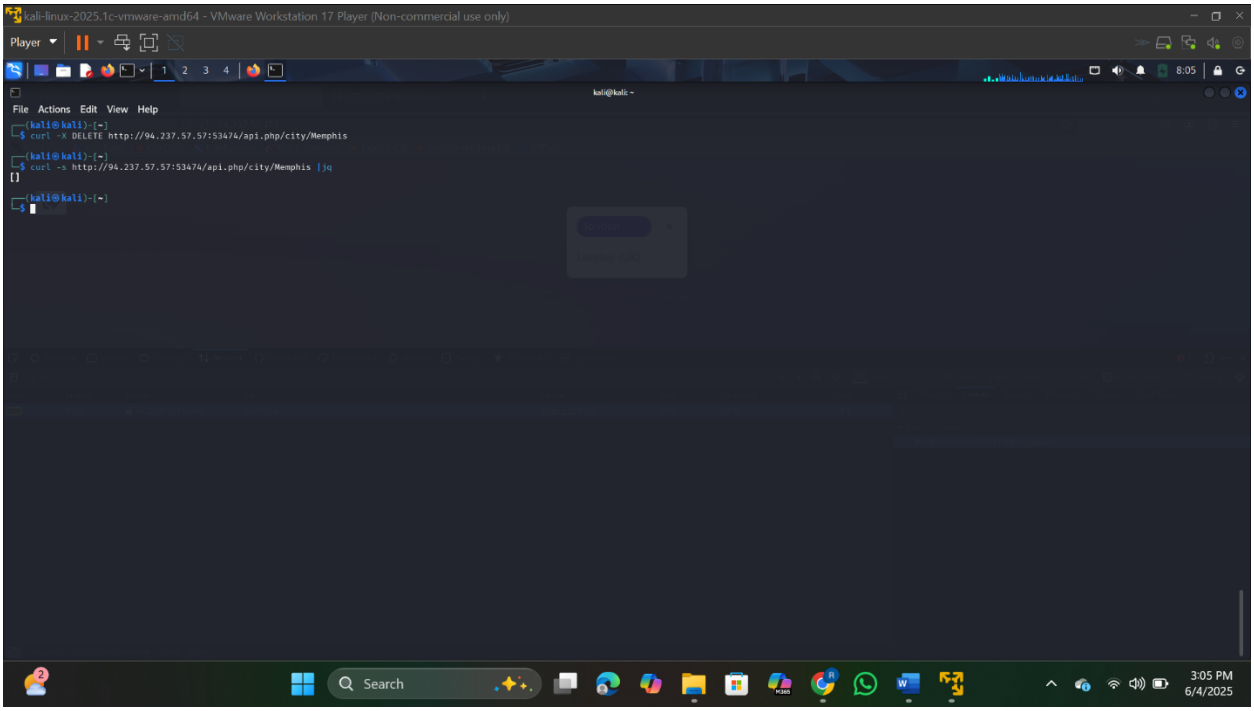

Fig 1.22 deleting city Memphis

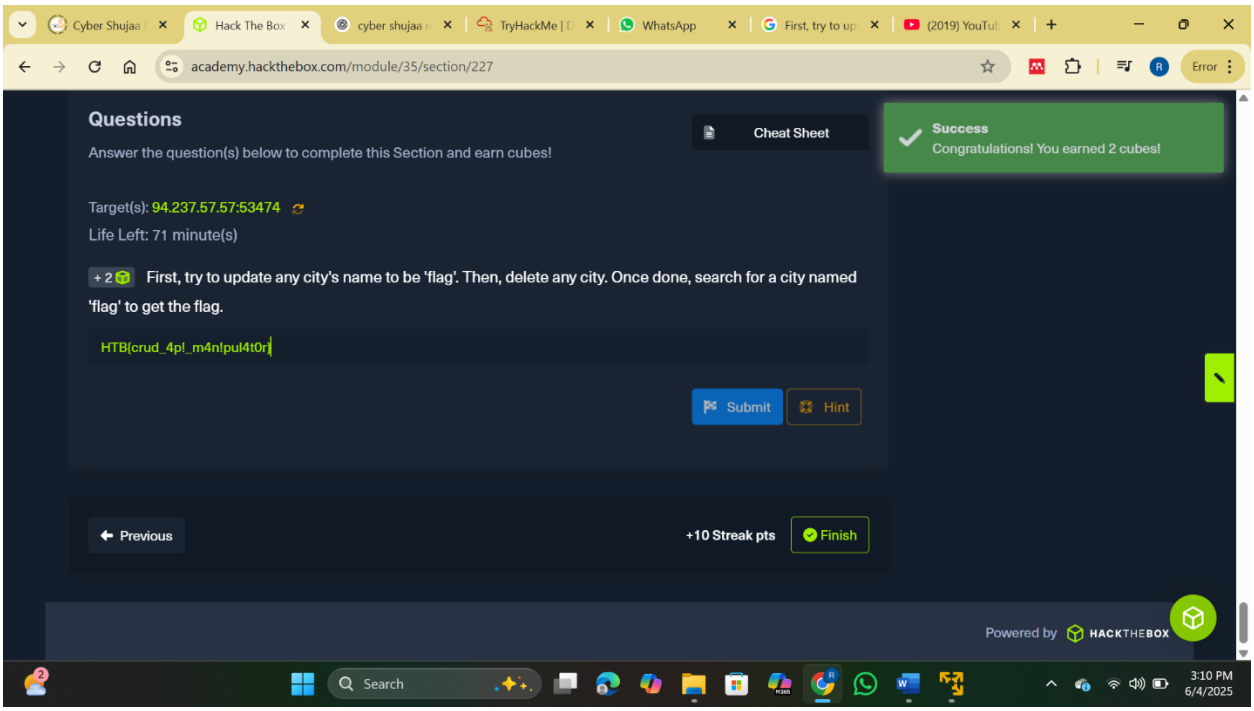

Fig 1.23 question on searching for a city named flag

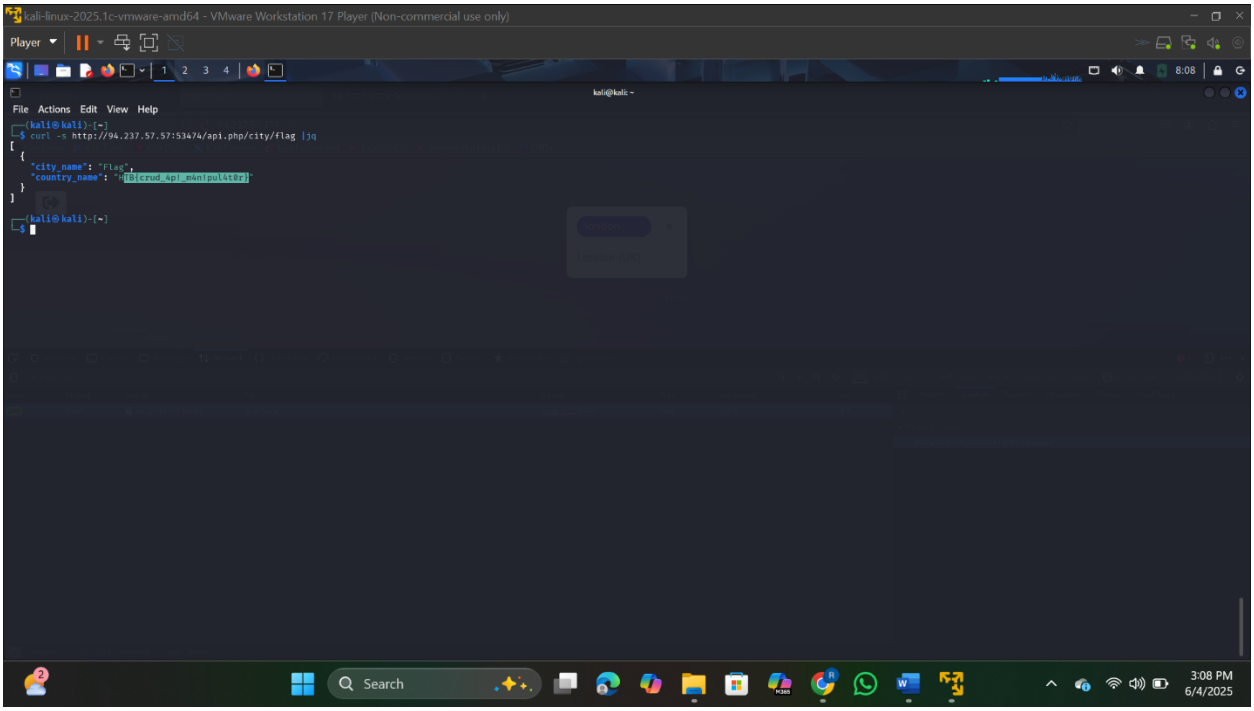

Fig 1.24 searching for a city named flag

Conclusion

Overall, this module provided a thorough and hands-on exploration of web requests, which are at the heart of both web functionality and web security. I gained practical experience in manually crafting and interpreting HTTP requests and responses, understanding the differences between encrypted and unencrypted communications, and interacting with web servers and APIs through cURL and browser-based tools. The ability to analyze request flows, extract authentication data, send custom headers, and manipulate API endpoints gave me the technical insight required for effective web application testing. Mastering these concepts is essential for anyone aspiring to secure, assess, or exploit web technologies, and this module laid the groundwork for more advanced web penetration testing techniques.