

**COURSE: CLOUD AND NETWORK SECURITY \_C1\_2025**

**STUDENT NAME: DIANA ROSE OGUDA**

**STUDENT NUMBER: CS-CNS09-25172**

**TUESDAY ,06 JUNE,2025**

**WEEK 3 ASSIGNMENT 2**

**HTB ACADEMY - WEB REQUESTS- ASSIGNMENT REPORT**

## 1. Introduction

This report documents my learning experience from the "Web Requests" module on Hack The Box Academy. The module introduces key concepts in how web clients and servers communicate over the internet. I explored the mechanics of HTTP and HTTPS, the structure of requests and responses, and how APIs function in web applications. Practical tools such as cURL and browser Developer Tools were used to interact with and analyze web traffic.

The knowledge gained from this module is foundational to anyone interested in web development, cybersecurity, or penetration testing, as it provides a clear understanding of how information is transferred and manipulated across the web.

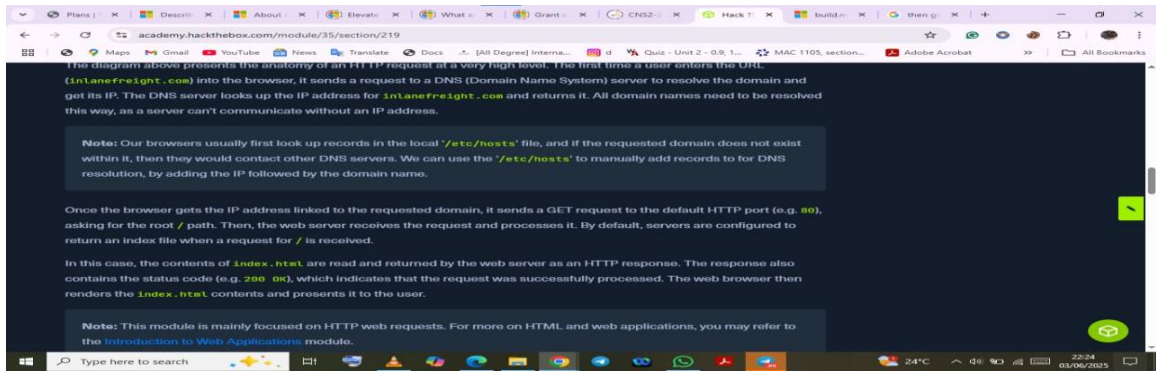
## 2. Module Experience and Learning Highlights

### HTTP and HTTPS

The module began with an overview of HTTP—the Hypertext Transfer Protocol—and its secure counterpart, HTTPS. I learned how HTTP facilitates data exchange between clients and servers, while HTTPS adds an encryption layer using TLS to secure this data. The differences between them were illustrated in terms of data confidentiality, integrity, and authentication, which are essential for securing web traffic.

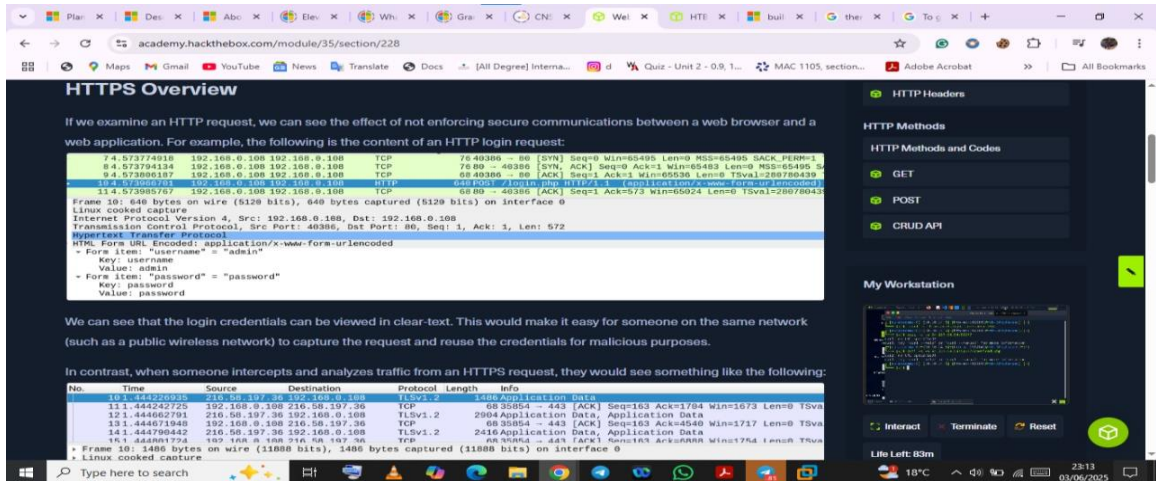
Component	Example	Description
Scheme	http://https://	This is used to identify the protocol being accessed by the client, and ends with a colon and a double slash (://).
User Info	admin:password@	This is an optional component that contains the credentials (separated by a colon :) used to

The diagram above presents the anatomy of an HTTP request at a very high level. The first time a user enters the URL (**inlanefreight.com**) into the browser, it sends a request to a DNS (Domain Name System) server to resolve the domain and get its IP. The DNS server looks up the IP address for **inlanefreight.com** and returns it. All domain names need to be resolved

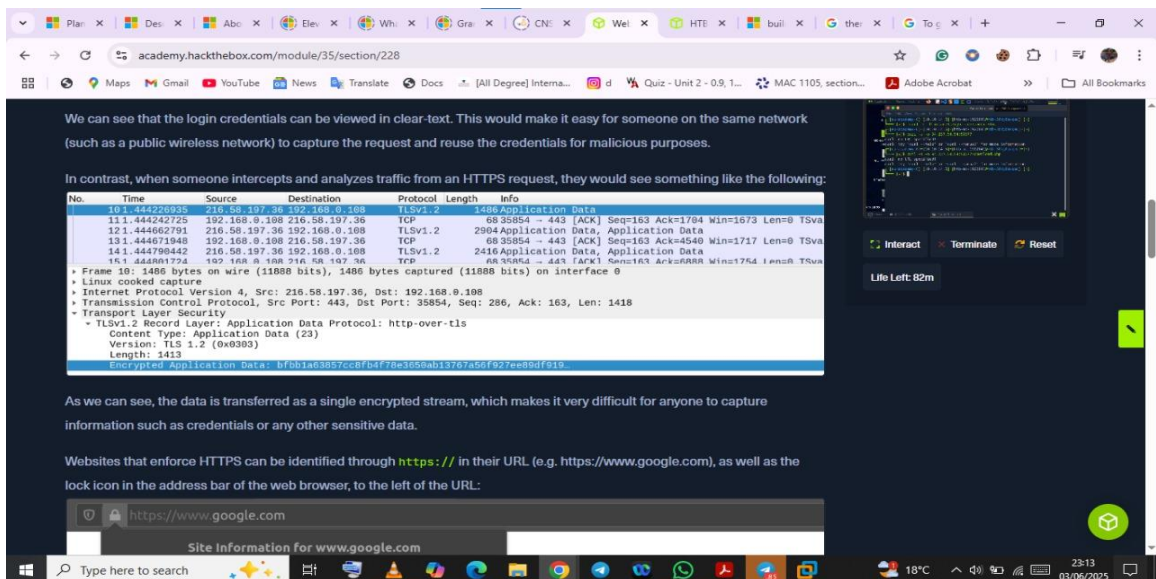


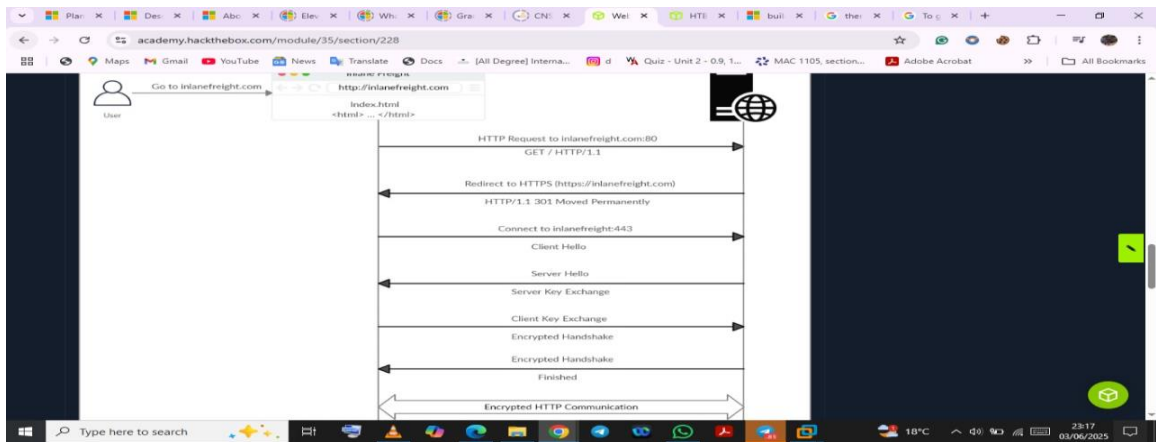
## HTTPS

### Seeing weakness of HTTP



## HTTPS added security





## cURL

Using the command-line tool cURL, I practiced sending requests to web servers without using a browser. This included GET, POST, PUT, and DELETE requests. These tasks demonstrated how to structure HTTP requests manually, analyze server responses, and test endpoints. It also gave insight into how attackers and security professionals use such tools during reconnaissance and testing.

academy.hackthebox.com/module/35/section/219

### cURL

In this module, we will be sending web requests through two of the most important tools for any web penetration tester, a Web Browser, like Chrome or Firefox, and the **cURL** command line tool.

**cURL** (client URL) is a command-line tool and library that primarily supports HTTP along with many other protocols. This makes it a good candidate for scripts as well as automation, making it essential for sending various types of web requests from the command line, which is necessary for many types of web penetration tests.

We can send a basic HTTP request to any URL by using it as an argument for cURL, as follows:

```
0gudai@ntb[/htb]$ curl inlanefreight.com
```

HyperText Transfer Protocol (HTTP)

```
<DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
...SNIP...
```

We see that cURL does not render the HTML/JavaScript/CSS code, unlike a web browser, but prints it in its raw format. However, as penetration testers, we are mainly interested in the request and response context, which usually becomes much faster and more convenient than a web browser.

academy.hackthebox.com/module/35/section/219

```
0gudai@ntb[/htb]$ curl -s -O inlanefreight.com/index.html
```

HyperText Transfer Protocol (HTTP)

This time, cURL did not print anything, as the output was saved into the **index.html** file. Finally, we may use the **-h** flag to see what other options we may use with cURL:

```
0gudai@ntb[/htb]$ curl -h
```

Usage: curl [options...] <url>

- d, --data <data> HTTP POST data
- h, --help <category> Get help for commands
- i, --include Include protocol response headers in the output
- o, --output <file> write to file instead of stdout
- O, --remote-name write output to a file named as the remote file
- s, --silent Silent mode
- u, --user <user:password> Server user and password
- A, --user-agent <name> Send User-Agent <name> to server
- v, --verbose Make the operation more talkative

This is not the full help, this menu is stripped into categories. Use "--help category" to get an overview of all categories. Use the user manual 'man curl' or the "--help all" flag for all options.



## Question

The screenshot shows a web browser window with the URL `academy.hackthebox.com/module/35/section/219`. The page is titled "Questions" and contains the following text:

Answer the question(s) below to complete this Section and earn cubes!

Target(s): 94.237.54.54:50577 🚩

Life Left: 59 minute(s)

+1 📖 To get the flag, start the above exercise, then use cURL to download the file returned by `/download.php` in the server shown above.

HTB{64\$!c\_cURL\_u\$3n}

Buttons: Submit, Hint, Mark Complete & Next

Powered by HACKTHEBOX

Integrated Terminal

## Answer

The screenshot shows a Kali Linux desktop environment with a terminal window open. The terminal output is as follows:

```
kali@kali: ~  
$ curl -o 94.237.54.54:50577/download.php  
curl: (2) no URL specified  
curl: try 'curl --help' or 'curl --manual' for more information  
$ curl http://94.237.54.54:50577/download.php  
HTB{64$!c_cURL_u$3n}
```

## cURL for HTTPS

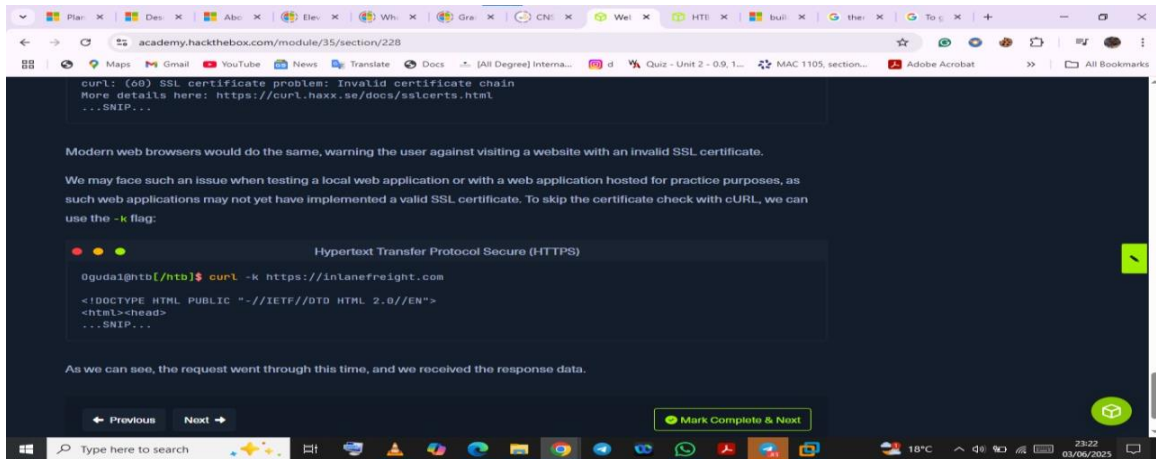
The screenshot shows a web browser window with the URL `academy.hackthebox.com/module/35/section/228`. The page is titled "cURL for HTTPS" and contains the following text:

cURL should automatically handle all HTTPS communication standards and perform a secure handshake and then encrypt and decrypt data automatically. However, if we ever contact a website with an invalid SSL certificate or an outdated one, then cURL by default would not proceed with the communication to protect against the earlier mentioned MITM attacks:

```
0guda1@htb[/htb]$ curl https://inlaneFreight.com  
curl: (60) SSL certificate problem: Invalid certificate chain  
More details here: https://curl.haxx.se/docs/sslcerts.html  
...SNIP...
```

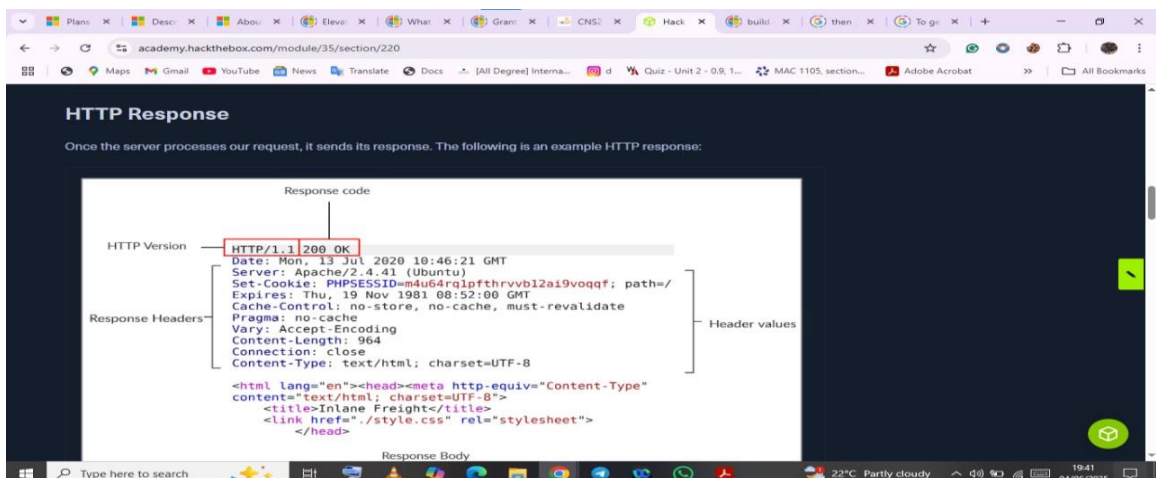
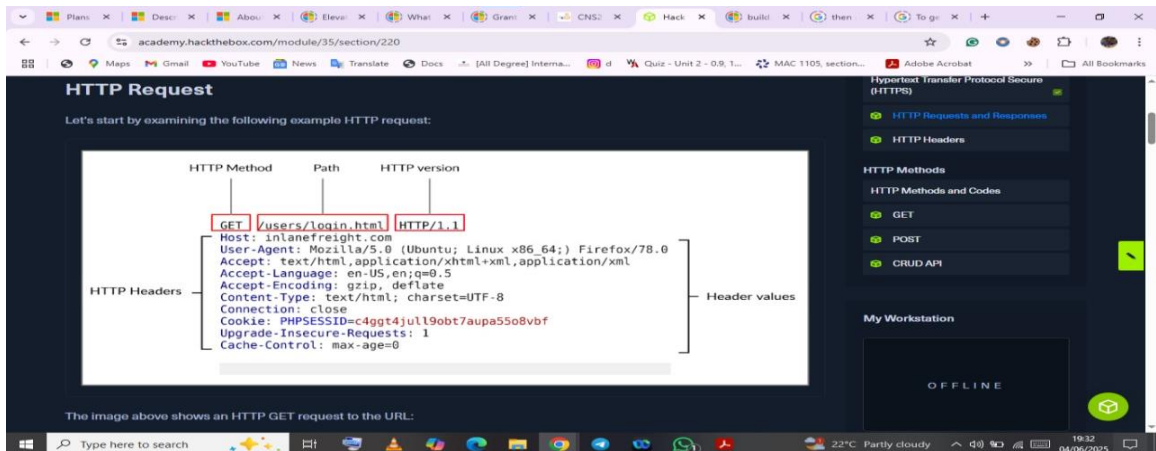
Modern web browsers would do the same, warning the user against visiting a website with an invalid SSL certificate.

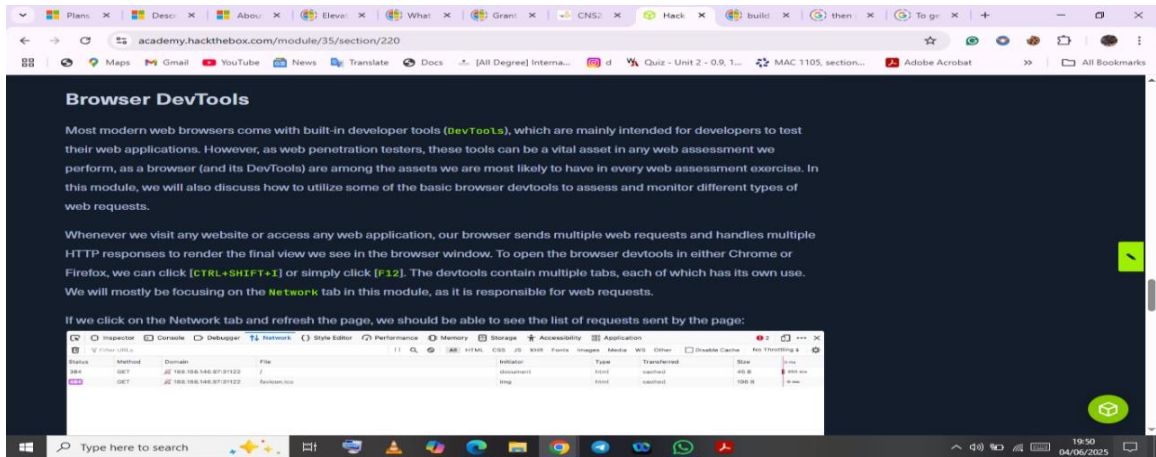
We may face such an issue when testing a local web application or with a web application hosted for practice purposes, as such web applications may not yet have implemented a valid SSL certificate. To skip the certificate check with cURL, we can use the `-k` flag:



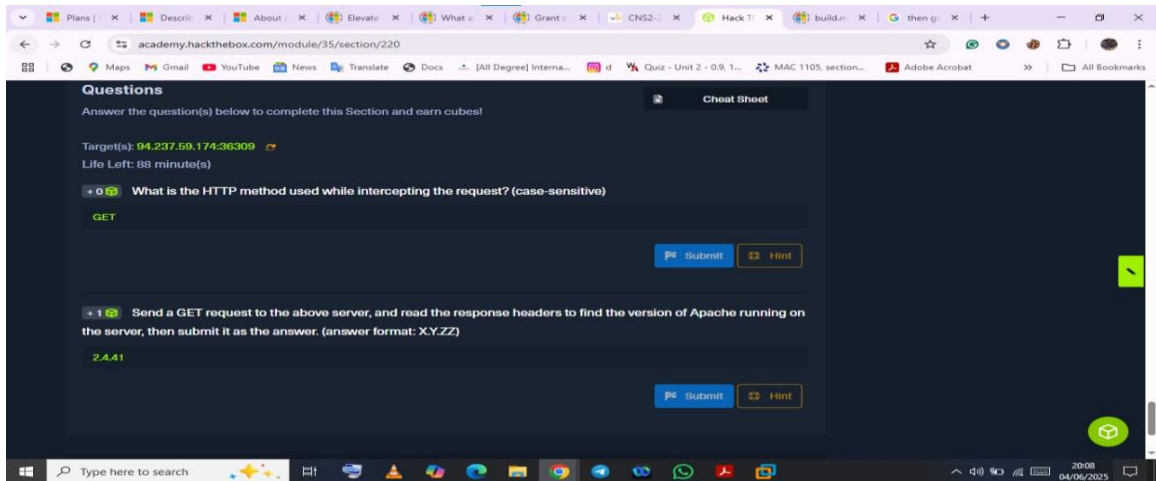
## HTTP Requests and Responses

This section covered the anatomy of an HTTP request and the structure of a server's response. I learned how browsers and tools like cURL generate requests containing methods, headers, URLs, and optional payloads, and how responses include status codes, headers, and content. Understanding this flow helped demystify the client-server communication model.

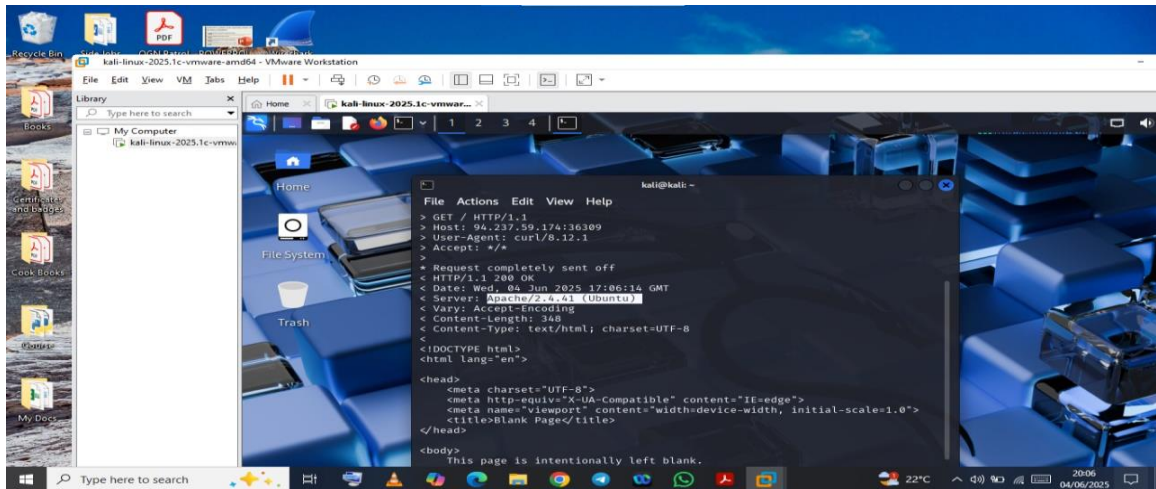




## Questions



## Answer





## HTTP Headers

Headers in both requests and responses play a crucial role in how data is exchanged and interpreted. I studied key headers like Content-Type, User-Agent, Authorization, Set-Cookie, and Cache-Control, understanding their purpose and implications. This section highlighted how headers can influence behavior, control access, and impact security.

The screenshot shows a web browser window with the URL `academy.hackthebox.com/module/35/section/223`. The page title is "Response Headers". The text explains that response headers are used in an HTTP response and do not relate to the content. It lists common headers: `Server`, `Set-Cookie`, and `WWW-Authenticate`. A table provides examples and descriptions for these headers.

Header	Example	Description
<code>Server</code>	<code>Server: Apache/2.2.14 (Ubuntu)</code>	Contains information about the HTTP server, which processed the request. It can be used to gain information about the server, such as its version, and estimate if further.
<code>Set-Cookie</code>	<code>Set-Cookie: PHPSESSID=b4e4fbd93540</code>	Contains the cookies needed for client identification. Browsers pass the cookies and store them for future requests. This header follows the same format as the <code>Cookie</code> request header.
<code>WWW-Authenticate</code>	<code>WWW-Authenticate: BASIC realm="/localhost"</code>	Notifies the client about the type of authentication required to access the requested resource.

Below the table, the section "Security Headers" is introduced, stating that with the increase in the variety of browsers and web-based attacks, defining certain headers is important.

The screenshot shows a web browser window with the URL `academy.hackthebox.com/module/35/section/223`. The page title is "HTTP Headers". The text explains that HTTP headers pass information between the client and the server. Some headers are only used with either requests or responses, while some other general headers are common to both. Headers can have one or multiple values, appended after the header name and separated by a colon. The page lists five categories of headers: 1. General Headers, 2. Entity Headers, 3. Request Headers, 4. Response Headers, and 5. Security Headers. A "Table of Contents" sidebar on the right lists various topics like "HyperText Transfer Protocol (HTTP)", "HTTP Methods and Codes", and "HTTP Headers".

The screenshot shows a web browser window with the URL `academy.hackthebox.com/module/35/section/223`. The page title is "General Headers". The text explains that general headers are used in both HTTP requests and responses. They are contextual and are used to describe the message rather than its contents. A table provides examples and descriptions for `Date` and `Connection` headers.

Header	Example	Description
<code>Date</code>	<code>Date: Wed, 10 Feb 2022 10:30:45 GMT</code>	Holds the date and time at which the message originated. It's preferred to convert the time to the standard UTC time zone.
<code>Connection</code>	<code>Connection: close</code>	Specifies if the current network connection should stay alive after the request finishes. Two commonly used values for this header are <code>close</code> and <code>keep-alive</code> . The <code>close</code> value from either the client or server means that they would like to terminate the connection, while the <code>keep-alive</code> header indicates that the connection should remain open to receive more data and input.

Below the table, the section "Entity Headers" is introduced, stating that entity headers can be common to both the request and response. These headers are used to describe the content (entity) transferred by a message. They are usually found in responses and POST or PUT requests.

The screenshot shows a web browser window with the URL `academy.hackthebox.com/module/35/section/223`. The page title is "Entity Headers". The text explains that entity headers can be common to both the request and response. These headers are used to describe the content (entity) transferred by a message. They are usually found in responses and POST or PUT requests. A table provides examples and descriptions for `Content-Type`, `Media-Type`, `Boundary`, `Content-Length`, and `Content-Encoding` headers.

Header	Example	Description
<code>Content-Type</code>	<code>Content-Type: text/html</code>	Used to describe the type of resource being transferred. The value is automatically added by the browsers on the client side and returned in the server response. The <code>charset</code> field denotes the encoding standard, such as UTF-8.
<code>Media-Type</code>	<code>Media-Type: application/pdf</code>	The <code>media-type</code> is similar to <code>Content-Type</code> , and describes the data being transferred. This header can play a crucial role in making the server interpret our input. The <code>charset</code> field may also be used with this header.
<code>Boundary</code>	<code>boundary="b4e4fbd93540"</code>	Acts as a marker to separate content when there is more than one in the same message. For example, within a form data, this boundary gets used as <code>--b4e4fbd93540</code> to separate different parts of the form.
<code>Content-Length</code>	<code>Content-Length: 385</code>	Holds the size of the entity being passed. This header is necessary as the server uses it to read data from the message body, and is automatically generated by the browser and tools like curl.
<code>Content-Encoding</code>	<code>Content-Encoding: gzip</code>	Data can undergo multiple transformations before being passed. For example, large amounts of data can be compressed to reduce the message size. The type of encoding being used should be specified using the <code>Content-Encoding</code> header.



academy.hackthebox.com/module/35/section/223

## Request Headers

The client sends **Request Headers** in an HTTP transaction. These headers are used in an HTTP request and do not relate to the content of the message. The following headers are commonly seen in HTTP requests.

Header	Example	Description
Host	Host: www.inlanefreight.com	Used to specify the host being queried for the resource. This can be a domain name or an IP address. HTTP servers can be configured to host different websites, which are revealed based on the hostname. This makes the host header an important enumeration target, as it can indicate the existence of other hosts on the target server.
User-Agent	User-Agent: curl/7.77.0	The <b>User-Agent</b> header is used to describe the client requesting resources. This header can reveal a lot about the client, such as the browser, its version, and the operating system.
Referer	Referer: http://www.inlanefreight.com/	Denotes where the current request is coming from. For example, clicking a link from Google search results would make <a href="https://google.com">https://google.com</a> the referer. Trusting this header can be dangerous as it can be easily manipulated, leading to unintended consequences.
Accept	Accept: */*	The <b>Accept</b> header describes which media types the client can understand. It can contain multiple media types separated by commas. The <b>*/*</b> value signifies that all media types are accepted.

academy.hackthebox.com/module/35/section/223

Accept	Accept: */*	The <b>Accept</b> header describes which media types the client can understand. It can contain multiple media types separated by commas. The <b>*/*</b> value signifies that all media types are accepted.
Cookie	Cookie: PHPSESSID=b4e4fbd93540	Contains cookie value pairs in the format <b>name=value</b> . A <b>cookie</b> is a piece of data stored on the client-side and on the server, which acts as an identifier. These are passed to the server per request, thus maintaining the client's access. Cookies can also serve other purposes, such as saving user preferences or session tracking. There can be multiple cookies in a single header separated by a semi-colon.
Authorization	Authorization: BASIC cGFzc3dvcmQK	Another method for the server to identify clients. After successful authentication, the server returns a token unique to the client. Unlike cookies, tokens are stored only on the client side and retrieved by the server per request. There are multiple types of authentication types based on the webserver and application type used.

A complete list of request headers and their usage can be found [here](#).

## Response Headers

**Response Headers** can be used in an HTTP response and do not relate to the content. Certain response headers such as **Age**, **Location**, and **Server** are used to provide more context about the response. The following headers are commonly seen in HTTP responses.

academy.hackthebox.com/module/35/section/223

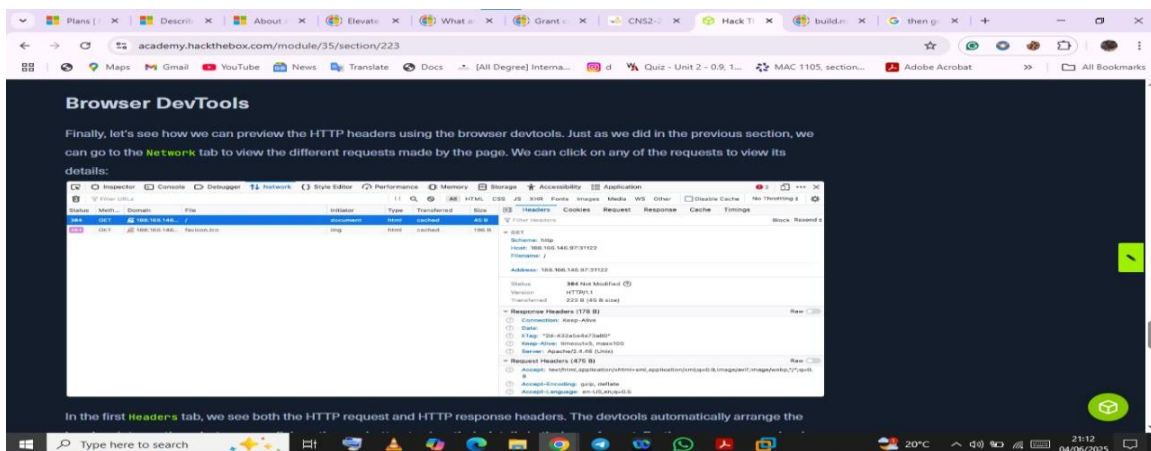
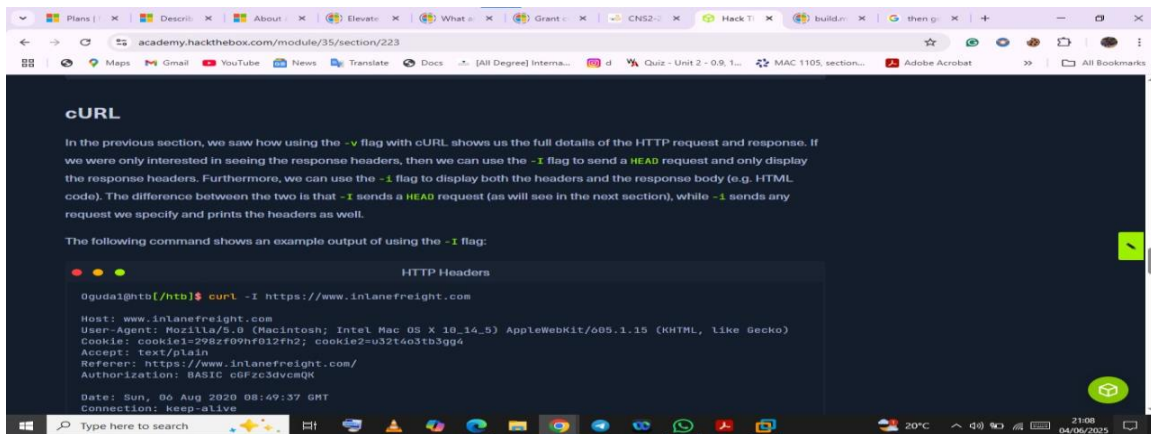
A complete list of request headers and their usage can be found [here](#).

## Response Headers

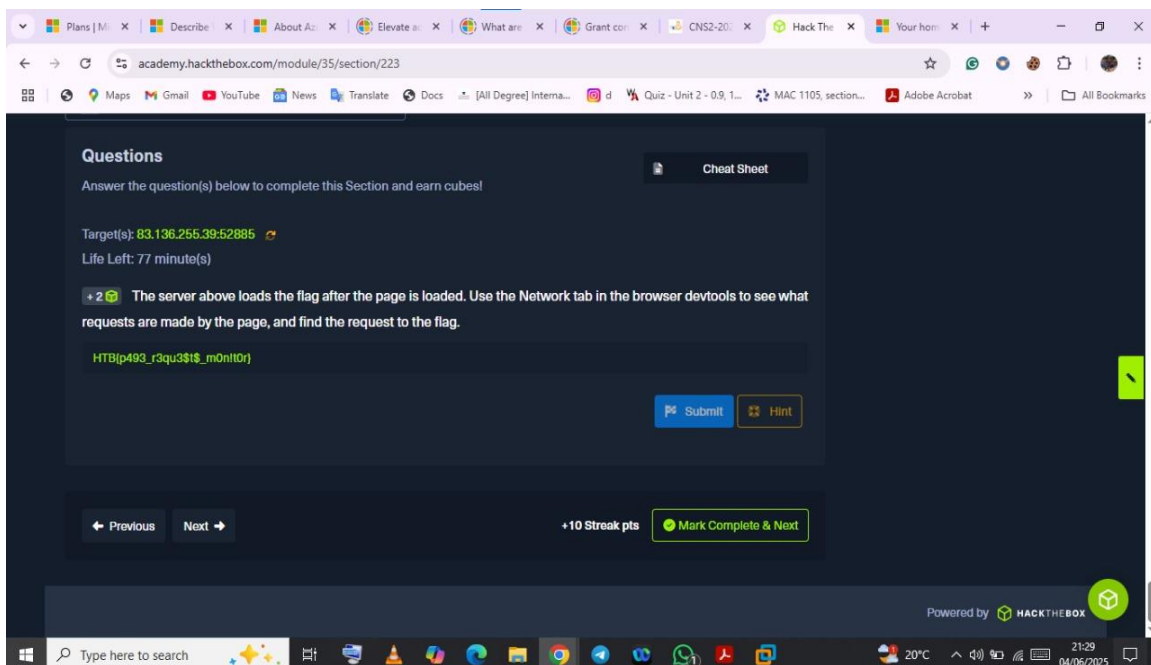
**Response Headers** can be used in an HTTP response and do not relate to the content. Certain response headers such as **Age**, **Location**, and **Server** are used to provide more context about the response. The following headers are commonly seen in HTTP responses.

Header	Example	Description
Server	Server: Apache/2.2.14 (Win32)	Contains information about the HTTP server, which processed the request. It can be used to gain information about the server, such as its version, and enumerate it further.
Set-Cookie	Set-Cookie: PHPSESSID=b4e4fbd93540	Contains the cookies needed for client identification. Browsers parse the cookies and store them for future requests. This header follows the same format as the <b>Cookie</b> request header.
WWW-Authenticate	WWW-Authenticate: BASIC realm="localhost"	Notifies the client about the type of authentication required to access the requested resource.

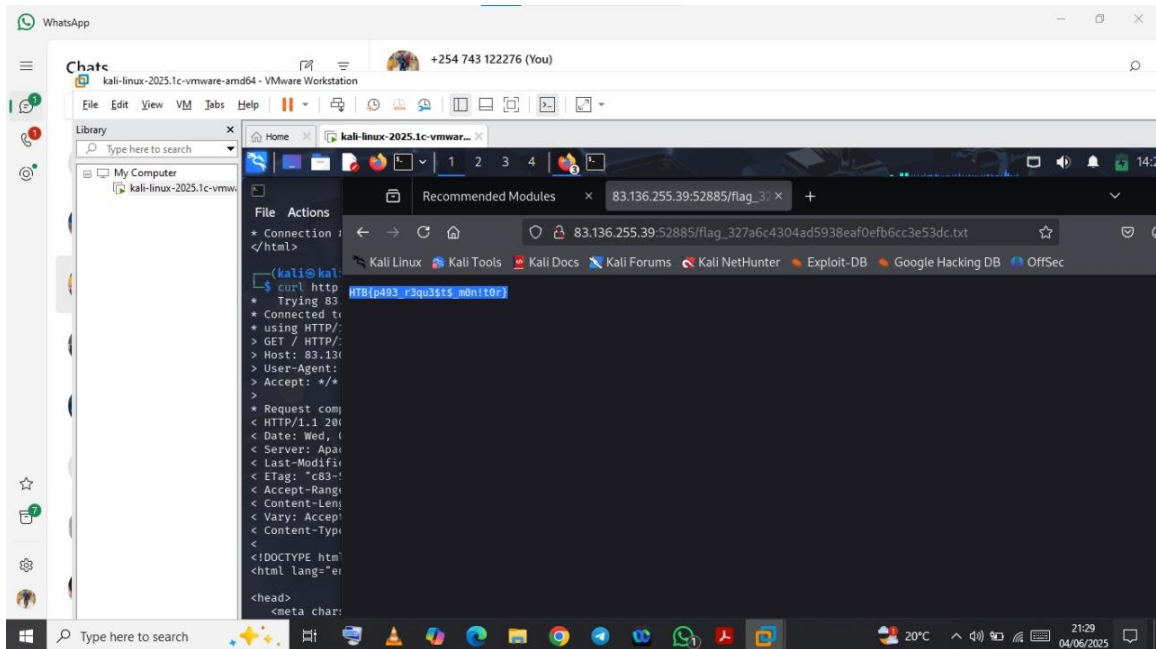
## Security Headers



## Question



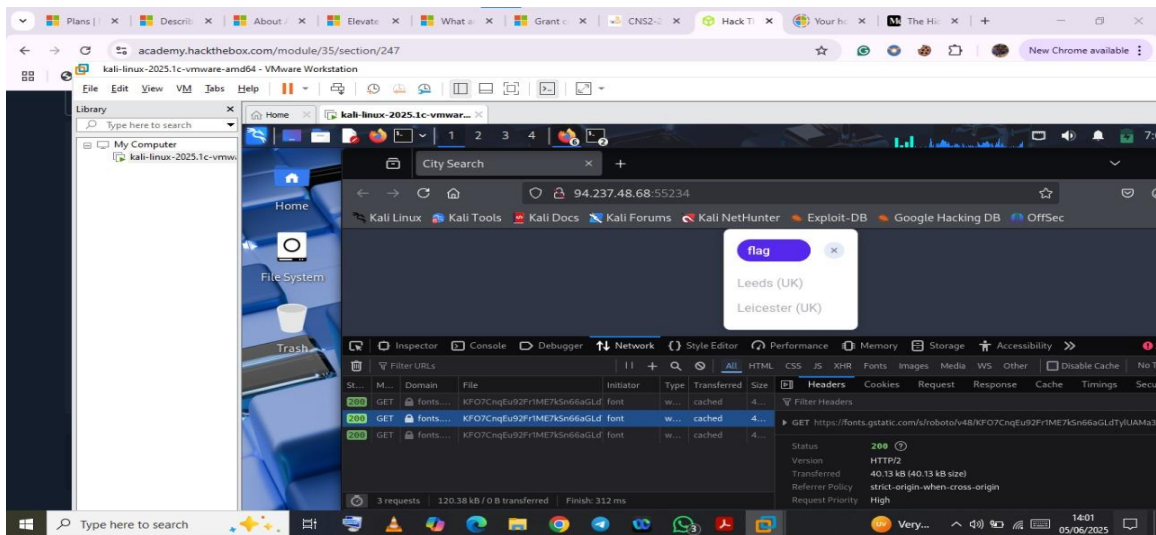
Answer



## HTTP Methods and Status Codes

This part of the module detailed the standard HTTP methods—GET, POST, PUT, DELETE—and how they are used in resource manipulation. Each method has a specific role in interacting with server resources. I also explored HTTP status codes, such as 200 (OK), 301 (Moved Permanently), 403 (Forbidden), 404 (Not Found), and 500 (Internal Server Error). Learning to interpret these codes is essential for debugging and understanding server behavior.

### 200 Code



## GET Requests

I learned that GET requests are used to retrieve data from a server without altering its state. Using both cURL and browser DevTools, I observed how GET requests are used to access resources such as HTML pages, images, and JSON responses from APIs. These requests typically include parameters in the URL and are idempotent, meaning they can be repeated without side effects.

## Question

The screenshot shows a web browser window with the URL `academy.hackthebox.com/module/35/section/247`. The page is titled "Questions" and contains the following text:

Answer the question(s) below to complete this Section and earn cubes!

Target(s): 94.237.48.68:55234 🚩

Life Left: 75 minute(s)

Authenticate to 94.237.48.68:55234 with user "admin" and password "admin"

🔍 The exercise above seems to be broken, as it returns incorrect results. Use the browser devtools to see what is the request it is sending when we search, and use cURL to search for 'flag' and obtain the flag.

HTB(curl\_g3773r)

Buttons: Submit, Hint

Progress: +10 Streak pts, Mark Complete & Next

## Answer

The screenshot shows a Kali Linux desktop environment with a terminal window open. The terminal output is as follows:

```
(kali@kali)-[~]
└─$ curl -v http://94.237.48.68:55234/search.php?search=flag -u admin:admin
* Trying 94.237.48.68:55234 ...
* Connected to 94.237.48.68 (94.237.48.68) port 55234
* using HTTP/1.x
* Server auth using Basic with user 'admin'
> GET /search.php?search=flag HTTP/1.1
* Host: 94.237.48.68:55234
* Authorization: Basic YWRtaW46YWRtaW4=
* User-Agent: curl/8.12.1
* Accept: */*
* Request completely sent off
< HTTP/1.1 200 OK
< Date: Thu, 05 Jun 2025 11:12:47 GMT
< Server: Apache/2.4.41 (Ubuntu)
< Cache-Control: no-cache, must-revalidate, max-age=0
< Content-Length: 25
< Content-Type: text/html; charset=UTF-8
<
Flag: HTB(curl_g3773r)
* Connection #0 to host 94.237.48.68 left intact
```



## POST Requests

POST requests are used to send data to a server to create or process resources. I practiced crafting POST requests to simulate form submissions and API calls. Unlike GET, POST includes the data in the request body rather than the URL and is used when the client intends to change the server state. I learned to view these requests and their payloads using DevTools' Network tab and to replicate them using cURL.

## Question

Questions

Answer the question(s) below to complete this Section and earn cubes!

Target(s): 83.136.253.207:51680

Life Left: 85 minute(s)

Authenticate to 83.136.253.207:51680 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{p0st\_r3p34t3r}

Submit Hint

Previous Next

Mark Complete & Next

Powered by HACKTHEBOX

## Answer

```
File Actions Edit View Help
</div>
<div class="search">
  <div class="bar">
    <div class="icon">
      <i></i>
    </div>
    <form>
      <input type="text">
    </form>
    <div class="close"></div>
    <ul id="results_list">
    </ul>
  </div>
  <em>Type a city name and hit <strong>Enter</strong></em>
</div>
</div>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<script src="/script.js"></script>
</body>
</html>

(kali@kali)-[~]
└─$ curl -X POST -d '{"search": "flag"}' -b 'PHPSESSID=bvnsnmknudiqtojmmdsh8ltoei' -H 'Content-Type: application/json' http://83.136.253.207:51680/search.php
{"flag": "HTB{p0st_r3p34t3r}"}
```

## CRUD Operations on APIs

The final segment focused on RESTful APIs and how CRUD operations—Create, Read, Update, Delete—are implemented using HTTP methods. I practiced interacting with API endpoints using POST for creating new resources, GET for reading data, PUT for updating existing resources, and DELETE for removing records. These tasks deepened my understanding of how web applications manage data behind the scenes and how structured, predictable HTTP communication powers API functionality.

### Question

Questions

Answer the question(s) below to complete this Section and earn cubes!

Target(s): 94.237.121.185:42859

Life Left: 53 minute(s)

+2 First, try to update any city's name to be 'flag'. Then, delete any city. Once done, search for a city named 'flag' to get the flag.

HTB(crud\_4pt\_m4npu4t0r)

Submit Hint

Previous +10 Streak pts Finish

Powered by HACKTHEBOX

Integrated Terminal

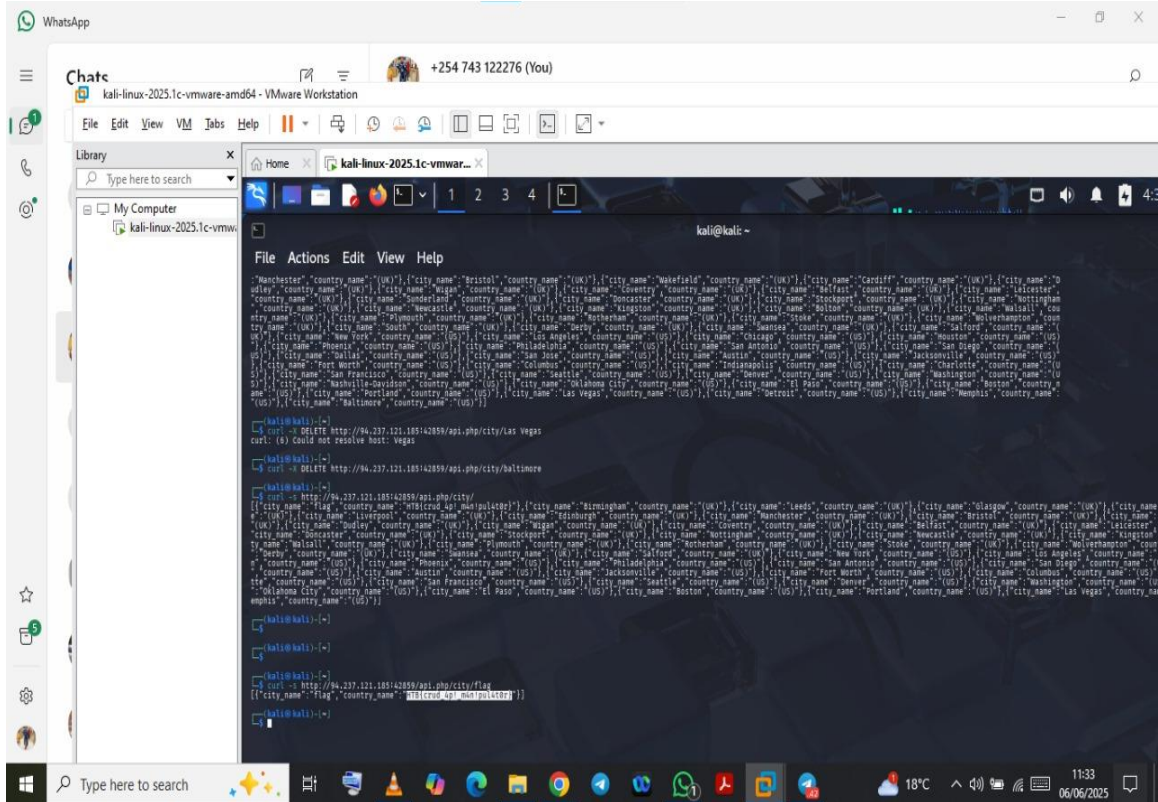
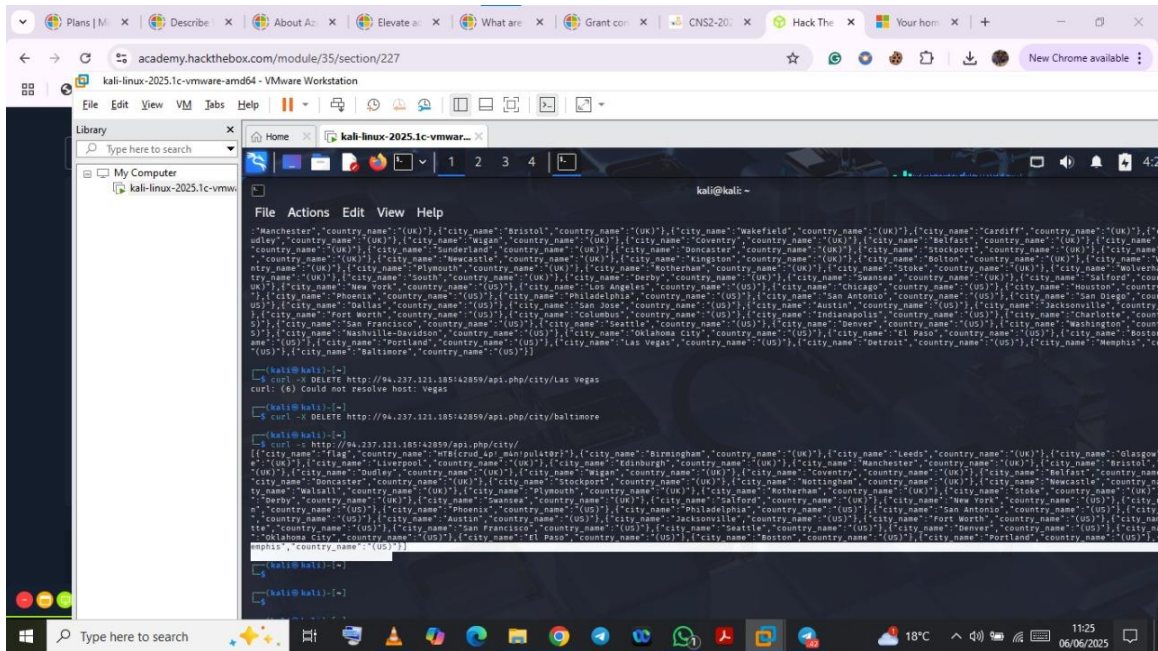
### Answer

```
kali@kali:~$ curl -X PUT http://94.237.121.185:42859/api.php/city/london -d '{"city_name":"flag","country_name":"UK"}' -H 'Content-Type: application/json'
```

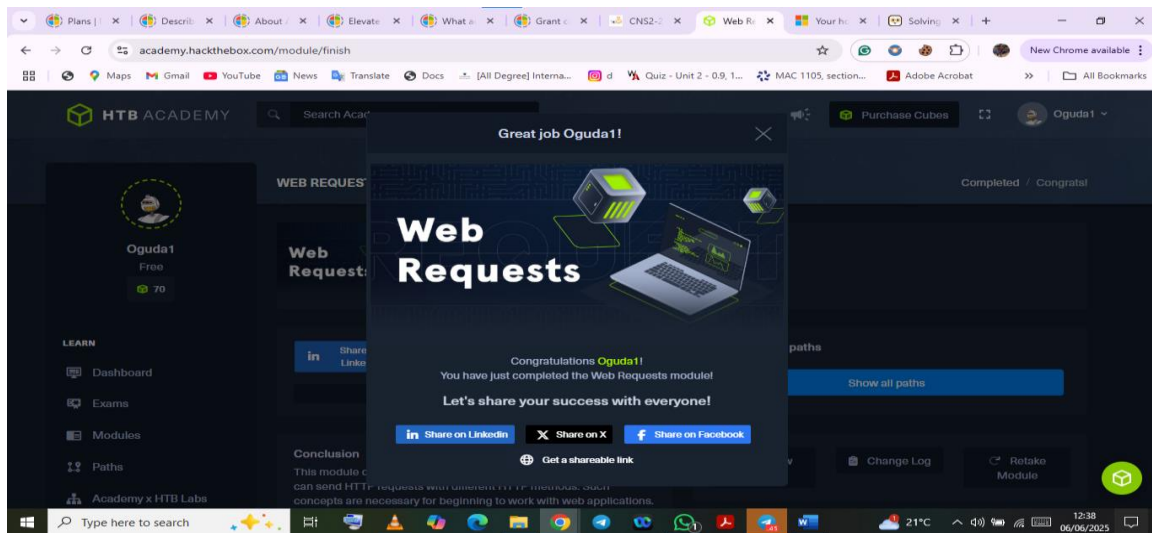
```
kali@kali:~$ curl -X DELETE http://94.237.121.185:42859/api.php/city/flag
```

```
kali@kali:~$ curl -X GET http://94.237.121.185:42859/api.php/city/flag
```

```
{
  "city_name": "flag",
  "country_name": "UK"
}
```



### 3. Completion Proof and Social Sharing



Shareable Link to Completion

<https://academy.hackthebox.com/achievement/1922141/35>

### 4. Conclusion

Completing the Web Requests module has been both insightful and rewarding. It offered a solid foundation in understanding how the web works from a technical perspective. I now feel more confident in analyzing HTTP traffic, using command-line tools to interact with APIs, and interpreting server responses. These skills are not only useful in development and testing environments but are also crucial for cybersecurity practices such as reconnaissance, vulnerability assessment, and penetration testing.

The hands-on exercises reinforced theoretical concepts and helped me develop a more analytical mindset when approaching web technologies. I look forward to building upon this knowledge in future modules and applying these skills in real-world security scenarios.