Week Three: Assignment 2: HTB Academy - Web Requests

CLOUD AND NETWORK SECURITY S2-2025

Cynthia W. Kamau

CS-CNS09-25020

Table of Contents

Introduction

This report outlines the key concepts and practical skills I gained from a foundational module on web application communication and security. It begins by exploring how modern applications interact with the internet through HTTP and HTTPS protocols. I examine the structure of HTTP requests and responses, the significance of status codes, headers, and methods such as GET, POST, PUT, and DELETE.

The report then covers how web browsers and DNS work together to resolve domain names into IP addresses, and how secure communication is established through HTTPS handshakes and SSL certificates. I also reflect on using tools like cURL for command-line web requests and Browser DevTools for inspecting and analyzing network traffic.

Additionally, the report touches on the importance of understanding API interactions—particularly CRUD operations—as part of web penetration testing. Each section draws from hands-on exercises and examples within the module, and concludes with the practical knowledge I've acquired to prepare for further cybersecurity learning and certifications.

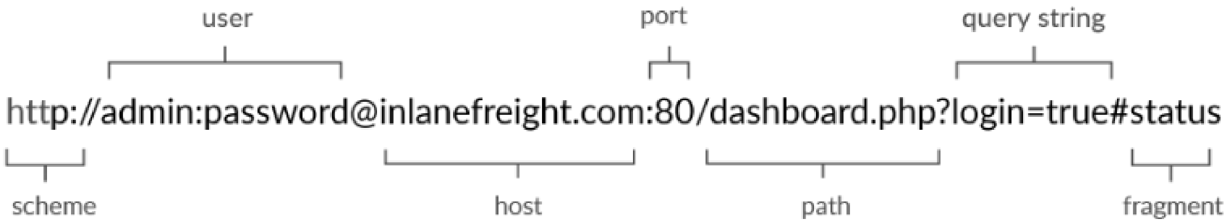
HyperText Transfer Protocol

Most modern web and mobile applications rely heavily on internet connectivity, with communication primarily happening through web requests using the HTTP protocol. HTTP, which operates at the application layer, is designed to access resources on the World Wide Web. It enables the exchange of hypertext—text that includes links to other content—in a way that users can easily navigate.

In a typical HTTP exchange, a client sends a request to a server, which then processes the request and responds with the appropriate resource. While HTTP usually runs over port 80, the port can be reconfigured depending on server settings. Every time a user visits a website by entering a URL (e.g., www.hackthebox.com), this type of HTTP request process takes place behind the scenes.

URL

Resources over HTTP are accessed via a URL, which offers many more specifications than simply specifying a website we want to visit. Let's look at the structure of a URL:


user
port
query string
http://admin:password@inlanefreight.com:80/dashboard.php?login=true#status
scheme
host
path
fragment

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 authenticate to the host, and is separated from the host with an at sign (@)
Host	inlanefreight.com	The host signifies the resource location. This can be a hostname or an IP address

Port	:80	The Port is separated from the Host by a colon (:). If no port is specified, http schemes default to port 80 and https default to port 443
Path	/dashboard.php	This points to the resource being accessed, which can be a file or a folder. If there is no path specified, the server returns the default index (e.g. index.html).
Query String	?login=true	The query string starts with a question mark (?), and consists of a parameter (e.g. login) and a value (e.g. true). Multiple parameters can be separated by an ampersand (&).
Fragments	#status	Fragments are processed by the browsers on the client-side to locate sections within the primary resource (e.g. a header or section on the page).

HTTP Flow

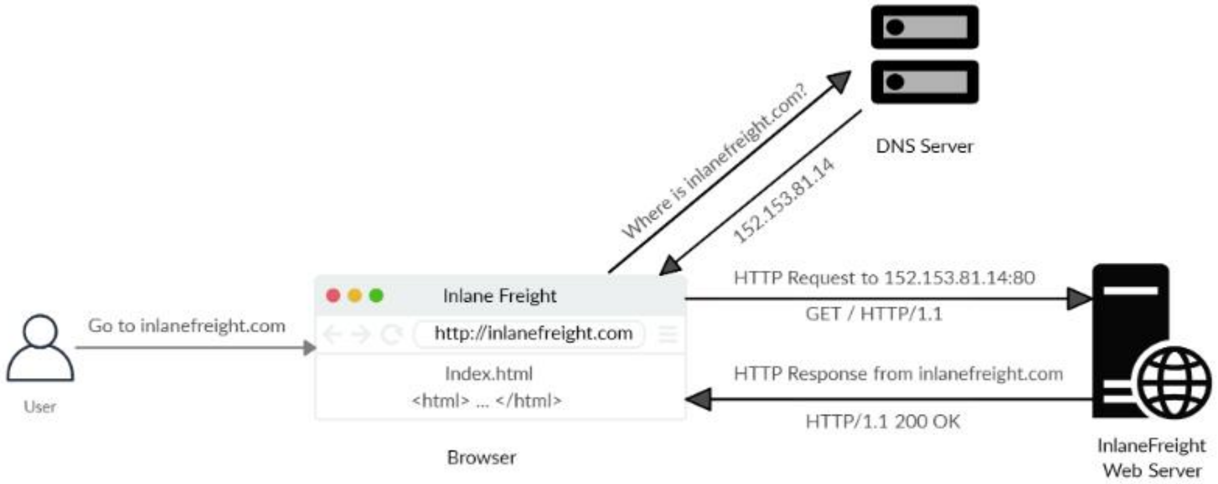


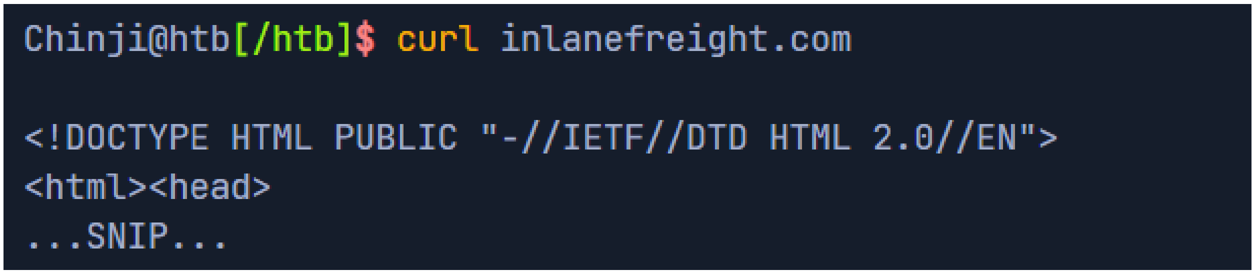
Figure 1: Anatomy of an HTTP request at a very high level

When a user enters a URL like ***inlanefreight.com*** into their browser for the first time;

- The browser sends a request to a DNS (Domain Name System) server to resolve the domain name into its corresponding IP address.
- Then the DNS server then retrieves and returns the IP address associated with the domain. *This step is essential because servers communicate using IP addresses, not domain names.*
- After the browser receives the IP address for the requested domain, it sends an HTTP GET request—usually to port 80—asking for the root path (/).
- The web server processes this request and, by default, serves the index file (typically *index.html*).
- This file is returned in an HTTP response, which also includes a status code such as *200 OK* to indicate success. The browser then renders the content of *index.html* and displays it to the user.

cURL

cURL (Client URL) is a powerful command-line tool and library used to send web requests. While it primarily supports HTTP, it also works with many other protocols. Its versatility makes it ideal for scripting and automation, which is especially useful during web penetration testing when sending different types of requests directly from the command line is required.

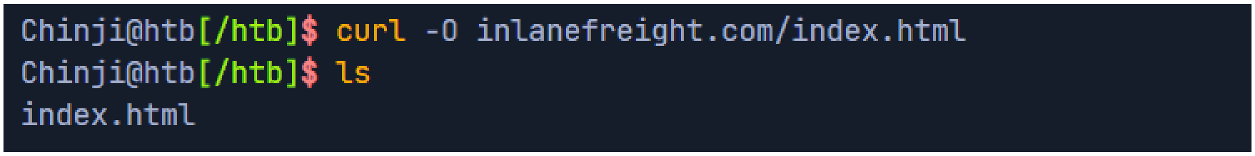


```
Chinji@htb[/htb]$ curl inlanefreight.com

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

Figure 2: Basic HTTP request to any URL by using an argument for cURL

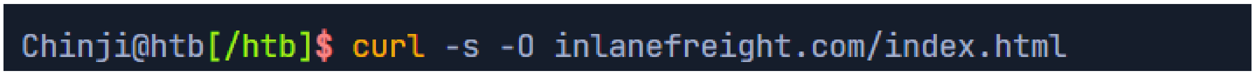
cURL does not render the HTML/JavaScript/CSS code, unlike a web browser, but prints it in its raw format. *However, as penetration testers, we are mainly interested in the request and response context, which usually becomes much faster and more convenient than a web browser.*

We may also use cURL to download a page or a file and output the content into a file using the -O flag. If we want to specify the output file name, we can use the -o flag and specify the name. Otherwise, we can use -O and cURL will use the remote file name, as follows:



```
Chinji@htb[/htb]$ curl -O inlanefreight.com/index.html
Chinji@htb[/htb]$ ls
index.html
```

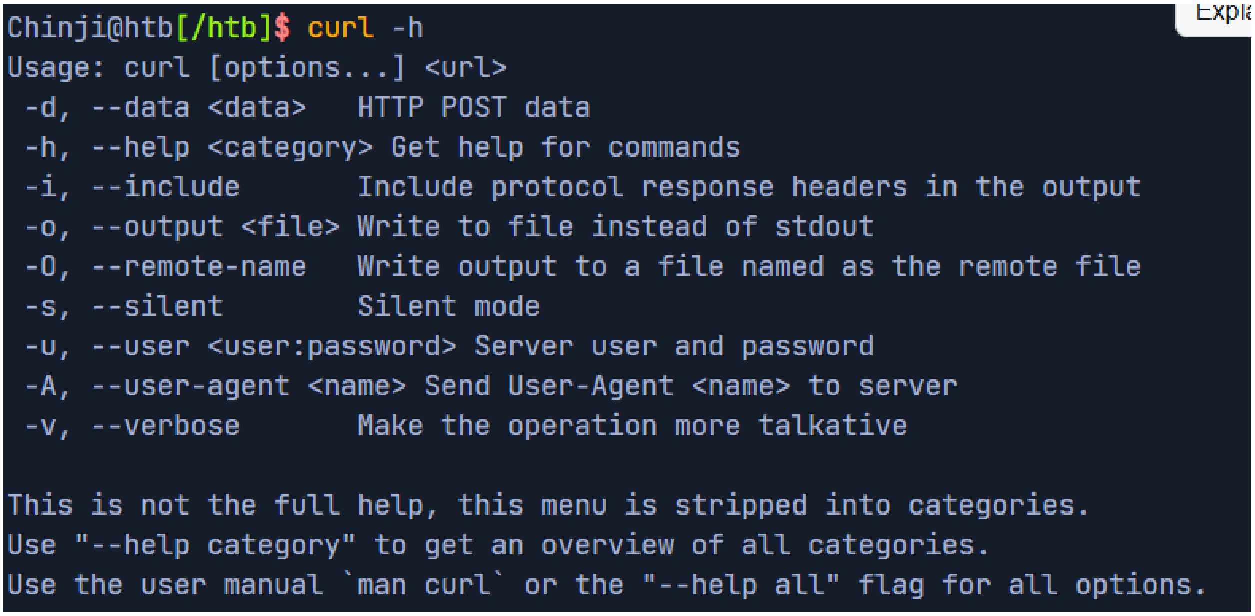
Figure 3: Downloading a page and saved into index.html- still prints some status while processing request



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

Figure 4: Silencing the status with the ***–s flag***

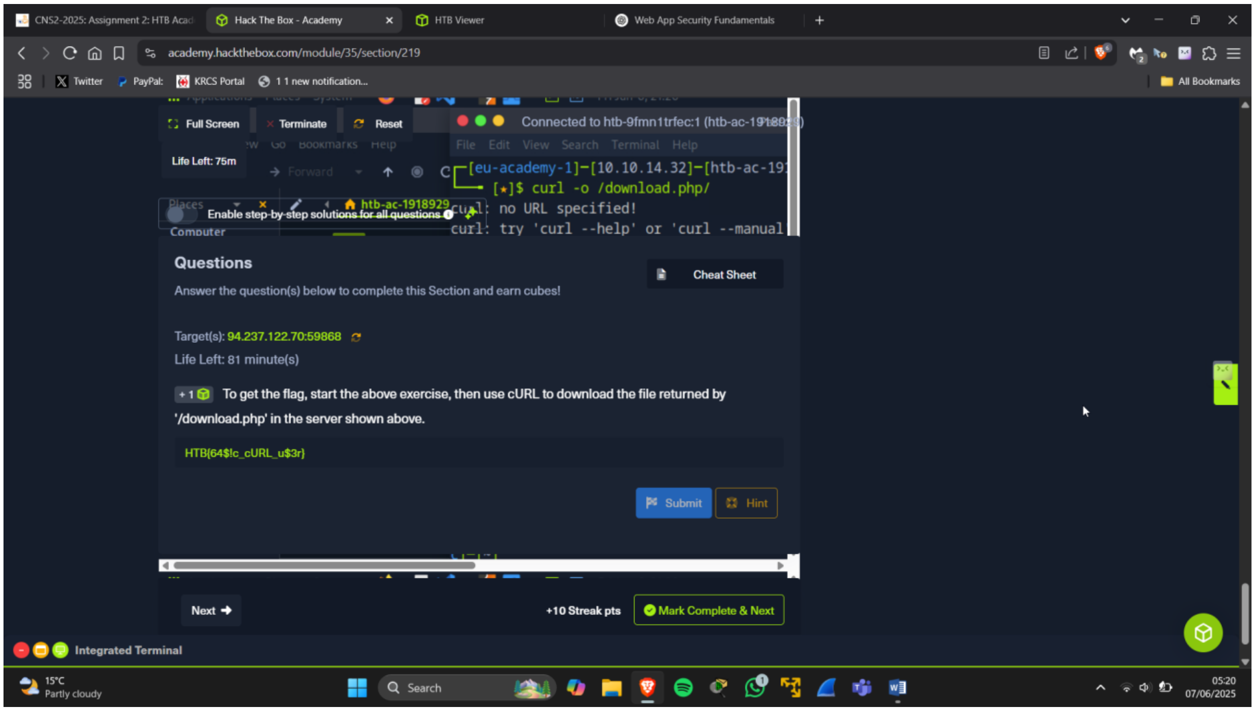
Week Three: Assignment 2: HTB Academy - Web Requests



```
Chinji@htb[/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.
```

*Figure 5: Using the **–h flag** to see other options to use with cURL*

*Note: We may use **man curl** to view the full cURL manual page*



```
Connected to htb-9fmn1trfec:1 (htb-ac-1918929)
┌─[eu-academy-1]─[10.10.14.32]─[htb-ac-191
└──╼ [★]$ curl -o /download.php/
curl: no URL specified!
curl: try 'curl --help' or 'curl --manual
```

Questions

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

Target(s): 94.237.122.70:59868

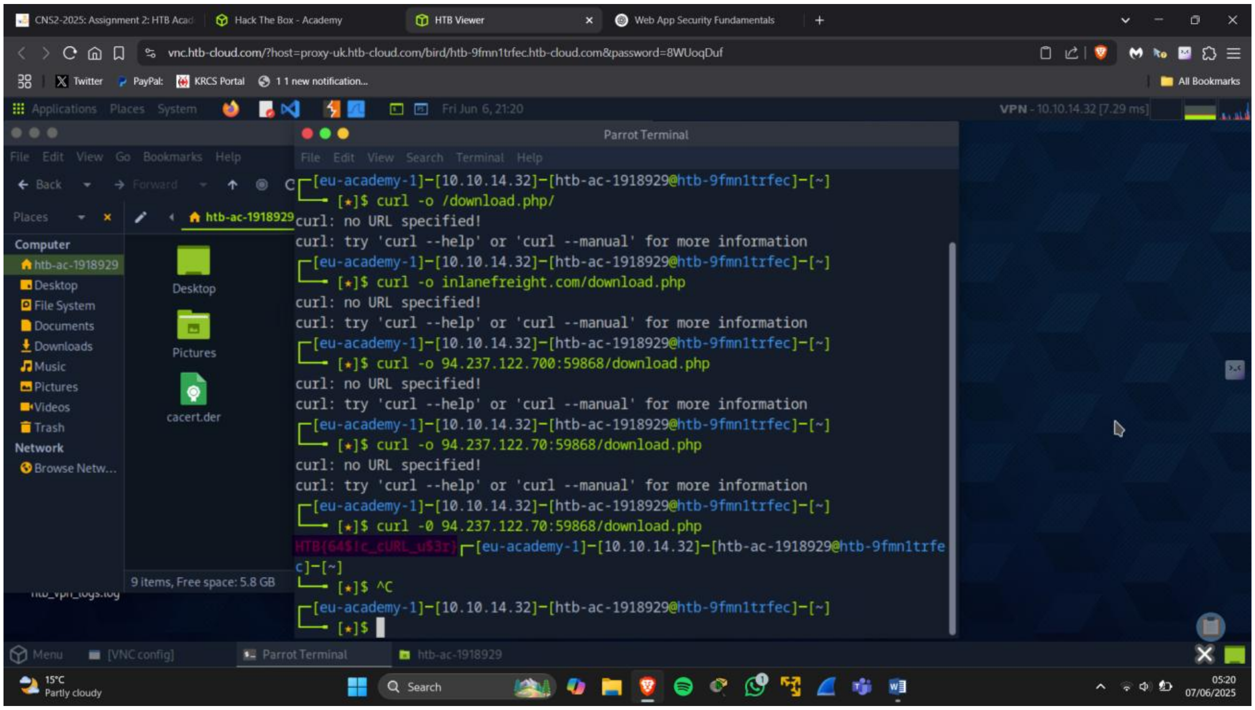
Life Left: 81 minute(s)

+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$3r}

Figure 6: Answer to module question

Week Three: Assignment 2: HTB Academy - Web Requests



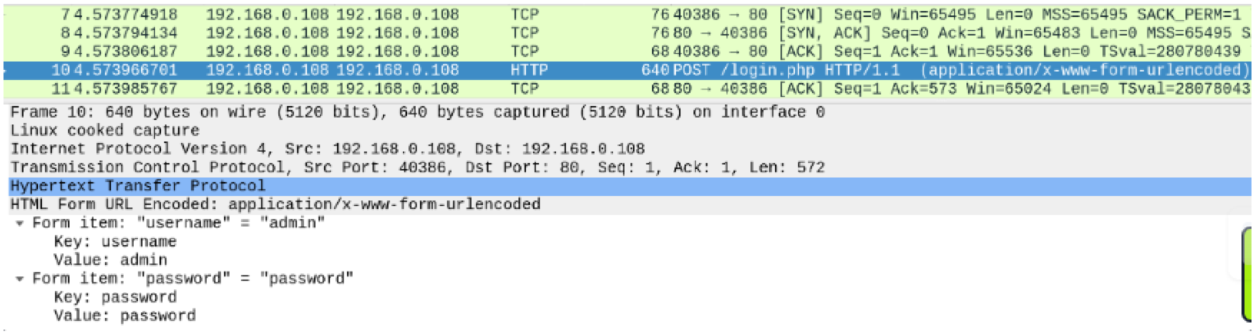
```
┌─[eu-academy-1]─[10.10.14.32]─[htb-ac-1918929@htb-9fmn1trfec]─[~]
└──╼ [★]$ curl -o /download.php/
curl: no URL specified!
curl: try 'curl --help' or 'curl --manual' for more information
┌─[eu-academy-1]─[10.10.14.32]─[htb-ac-1918929@htb-9fmn1trfec]─[~]
└──╼ [★]$ curl -o inlanefreight.com/download.php
curl: no URL specified!
curl: try 'curl --help' or 'curl --manual' for more information
┌─[eu-academy-1]─[10.10.14.32]─[htb-ac-1918929@htb-9fmn1trfec]─[~]
└──╼ [★]$ curl -o 94.237.122.700:59868/download.php
curl: no URL specified!
curl: try 'curl --help' or 'curl --manual' for more information
┌─[eu-academy-1]─[10.10.14.32]─[htb-ac-1918929@htb-9fmn1trfec]─[~]
└──╼ [★]$ curl -o 94.237.122.70:59868/download.php
curl: no URL specified!
curl: try 'curl --help' or 'curl --manual' for more information
┌─[eu-academy-1]─[10.10.14.32]─[htb-ac-1918929@htb-9fmn1trfec]─[~]
└──╼ [★]$ curl -0 94.237.122.70:59868/download.php
HTB{64$!c_cURL_u$3r}┌─[eu-academy-1]─[10.10.14.32]─[htb-ac-1918929@htb-9fmn1trfe
c]─[~]
└──╼ [★]$ ^C
┌─[eu-academy-1]─[10.10.14.32]─[htb-ac-1918929@htb-9fmn1trfec]─[~]
└──╼ [★]$ 
```

Figure 7: Evidence to support answer above

HyperText Transfer Protocol Secure (HTTPS)

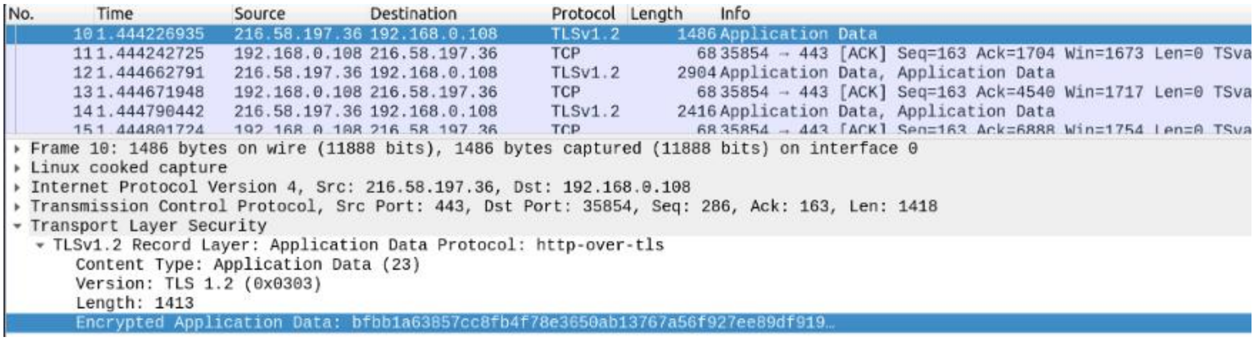
HTTPS (Hypertext Transfer Protocol Secure) was developed to ensure that all data exchanged between the client and server is encrypted. This encryption prevents third parties from accessing or reading intercepted data. Due to its security benefits, HTTPS has become the standard for most websites, and HTTP is gradually being phased out. In fact, many modern browsers are beginning to block access to non-secure HTTP sites altogether.

HTTPS Overview



```
  7 4.573774918   192.168.0.108 192.168.0.108   TCP      76 40386 → 80 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1
  8 4.573794134   192.168.0.108 192.168.0.108   TCP      76 80 → 40386 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 S
  9 4.573806187   192.168.0.108 192.168.0.108   TCP      68 40386 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=280780439
 10 4.573966701   192.168.0.108 192.168.0.108   HTTP    640 POST /login.php HTTP/1.1  (application/x-www-form-urlencoded)
 11 4.573985767   192.168.0.108 192.168.0.108   TCP      68 80 → 40386 [ACK] Seq=1 Ack=573 Win=65024 Len=0 TSval=28078043
Frame 10: 640 bytes on wire (5120 bits), 640 bytes captured (5120 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 192.168.0.108, Dst: 192.168.0.108
Transmission Control Protocol, Src Port: 40386, Dst Port: 80, Seq: 1, Ack: 1, Len: 572
Hypertext Transfer Protocol
HTML Form URL Encoded: application/x-www-form-urlencoded
▾ Form item: "username" = "admin"
    Key: username
    Value: admin
▾ Form item: "password" = "password"
    Key: password
    Value: password
```

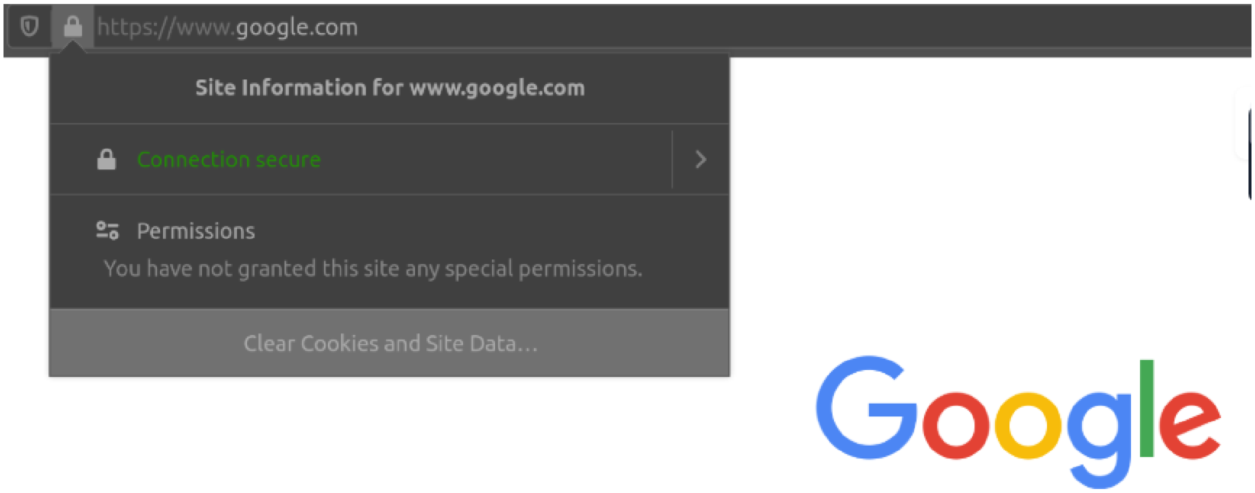
Figure 8: Login credentials seen in clear text via HTTP



```
No.   Time          Source        Destination   Protocol Length Info
 10 1.444226935   216.58.197.36 192.168.0.108   TLSv1.2    1486 Application Data
 11 1.444242725   192.168.0.108 216.58.197.36   TCP          68 35854 → 443 [ACK] Seq=163 Ack=1704 Win=1673 Len=0 TSva
 12 1.444662791   216.58.197.36 192.168.0.108   TLSv1.2    2904 Application Data, Application Data
 13 1.444671948   192.168.0.108 216.58.197.36   TCP          68 35854 → 443 [ACK] Seq=163 Ack=4540 Win=1717 Len=0 TSva
 14 1.444790442   216.58.197.36 192.168.0.108   TLSv1.2    2416 Application Data, Application Data
 15 1.444801724   192.168.0.108 216.58.197.36   TCP          68 35854 → 443 [ACK] Seq=163 Ack=6888 Win=1754 Len=0 TSva
▸ Frame 10: 1486 bytes on wire (11888 bits), 1486 bytes captured (11888 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 216.58.197.36, Dst: 192.168.0.108
▸ Transmission Control Protocol, Src Port: 443, Dst Port: 35854, Seq: 286, Ack: 163, Len: 1418
▾ Transport Layer Security
  ▾ TLSv1.2 Record Layer: Application Data Protocol: http-over-tls
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 1413
      Encrypted Application Data: bfbb1a63857cc8fb4f78e3650ab13767a56f927ee89df919…
```

Figure 9: Encrypted stream via HTTPS

Websites that enforce HTTPS can be identified through **https:// in their URL** (e.g. https://www.google.com), as well as the lock icon in the address bar of the web browser, to the left of the URL:



https://www.google.com

Site Information for www.google.com

Connection secure

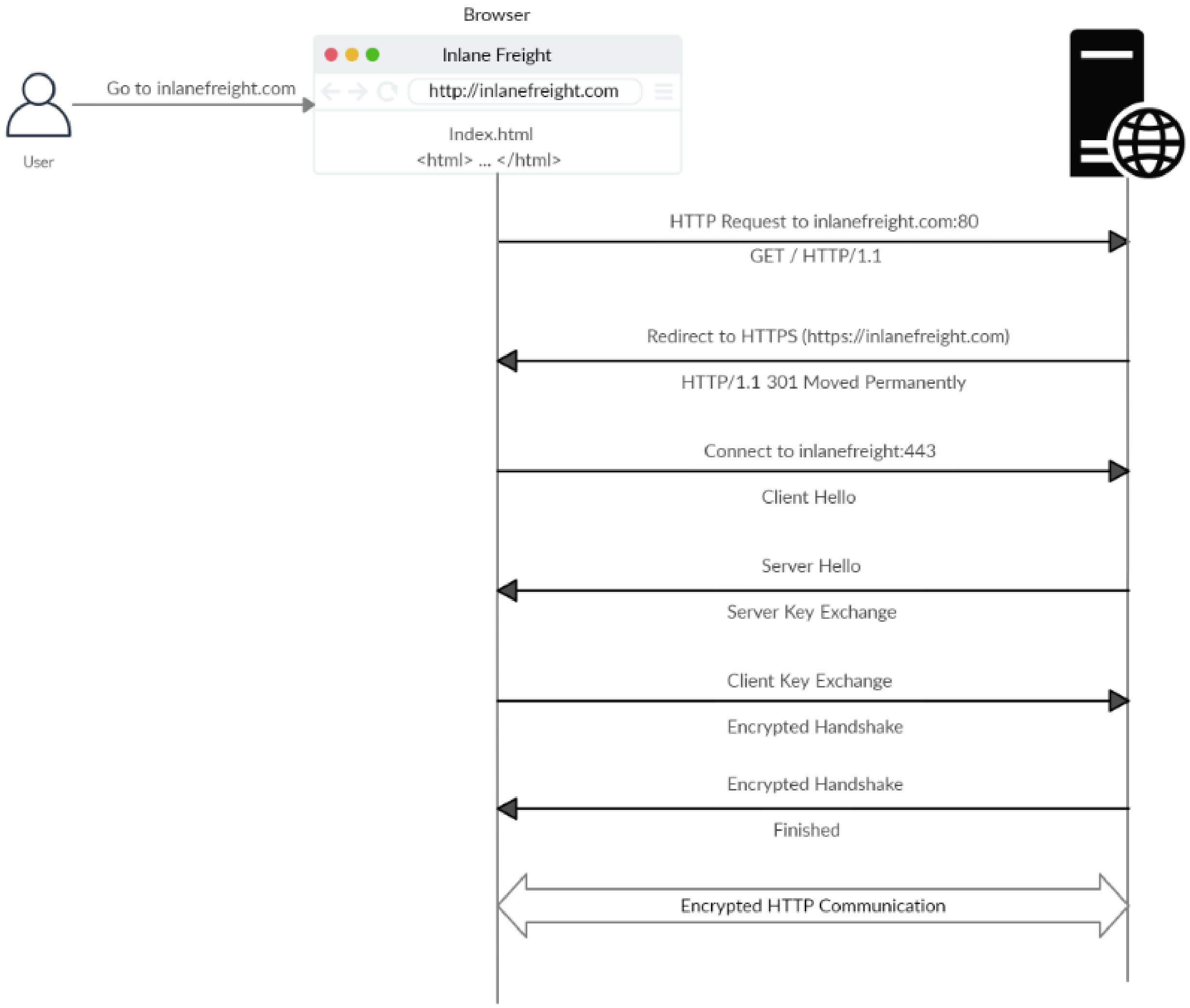
Permissions

You have not granted this site any special permissions.

Figure 10: HTTPS makes Google traffic to be encrypted

***Note**: Although the data transferred through the HTTPS protocol may be encrypted, the request may still reveal the visited URL if it contacted a clear-text DNS server. For this reason, it is recommended to utilize encrypted DNS servers (e.g. 8.8.8.8 or 1.1.1.1), or utilize a VPN service to ensure all traffic is properly encrypted.*

HTTPS Flow



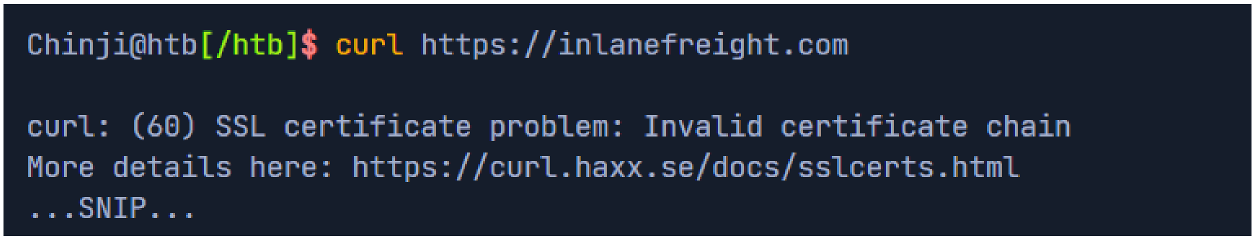
- If a user types `http://` instead of `https://`, the browser first sends a request to port 80 (HTTP).
- The server detects this and responds with a **301 Moved Permanently** status, redirecting to port 443 (HTTPS).
- The browser then begins the **SSL/TLS handshake**:
  - Sends a **"Client Hello"** message with browser info.
  - Server responds with **"Server Hello"** and shares its SSL certificate.
  - A **key exchange** occurs for secure communication.
  - The client verifies the certificate and responds with its own part of the handshake.
- Once the handshake is successful, **encrypted HTTP communication** begins over HTTPS.

This process ensures secure data transfer and prevents interception.

***Note**: Depending on the circumstances, an attacker may be able to perform an HTTP downgrade attack, which downgrades HTTPS communication to HTTP, making the data transferred in clear-text. This is done by setting up a Man-In-The-Middle (MITM) proxy to transfer all traffic through the attacker's host* *without the user's knowledge. However, most modern browsers, servers, and web applications protect against this attack.*

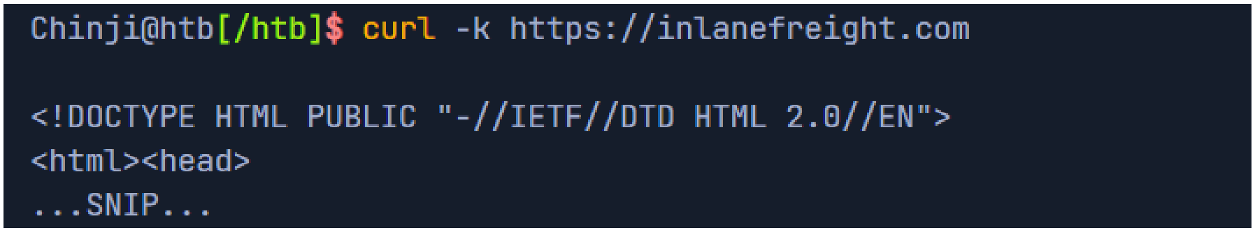
cURL for HTTPS

cURL automatically manages HTTPS communication, including performing the SSL/TLS handshake and handling encryption and decryption of data. This process is seamless and ensures secure communication. *However, if cURL encounters a website with an invalid or expired SSL certificate, it will stop the connection by default. This built-in behavior is a security measure to prevent man-in-the-middle (MITM) attacks and maintain the integrity of the data being transmitted*.



```
Chinji@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...
```

Figure 11: Showing the invalid SSL certificate warning the user against visiting the website



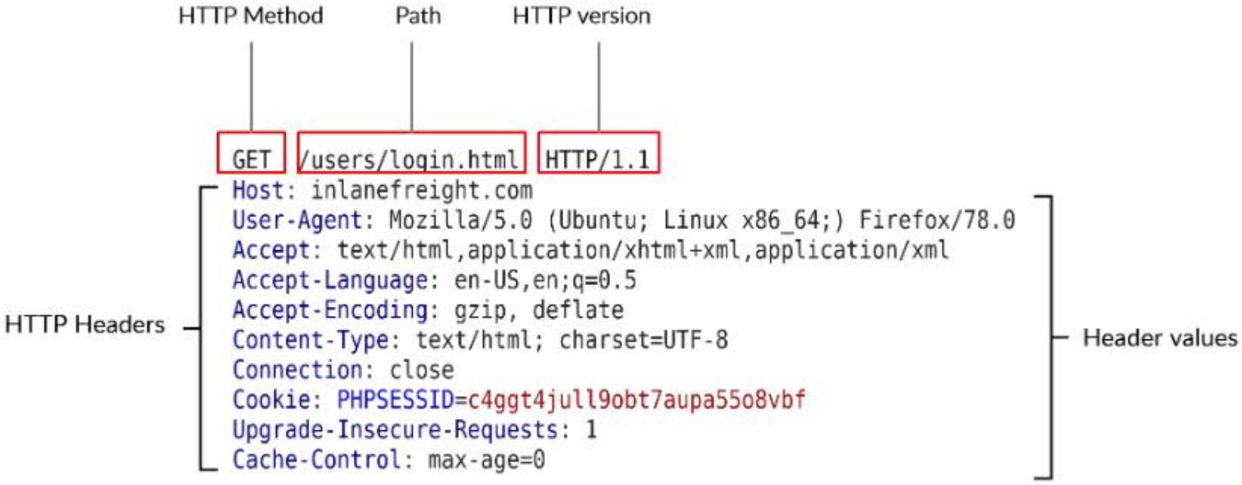
```
Chinji@htb[/htb]$ curl -k https://inlanefreight.com

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

Figure 12: Skipping the certificate check with –k flag

HTTP Requests and Responses

HTTP communication involves two main parts: **a request from the client and a response from the server**. The client (like a browser or cURL) sends an HTTP request that includes details such as the URL, path, parameters, headers, and any additional data. The server receives this request, processes it, and returns an HTTP response. This response includes a status code and, if permitted, the requested resource or data.

HTTP Request



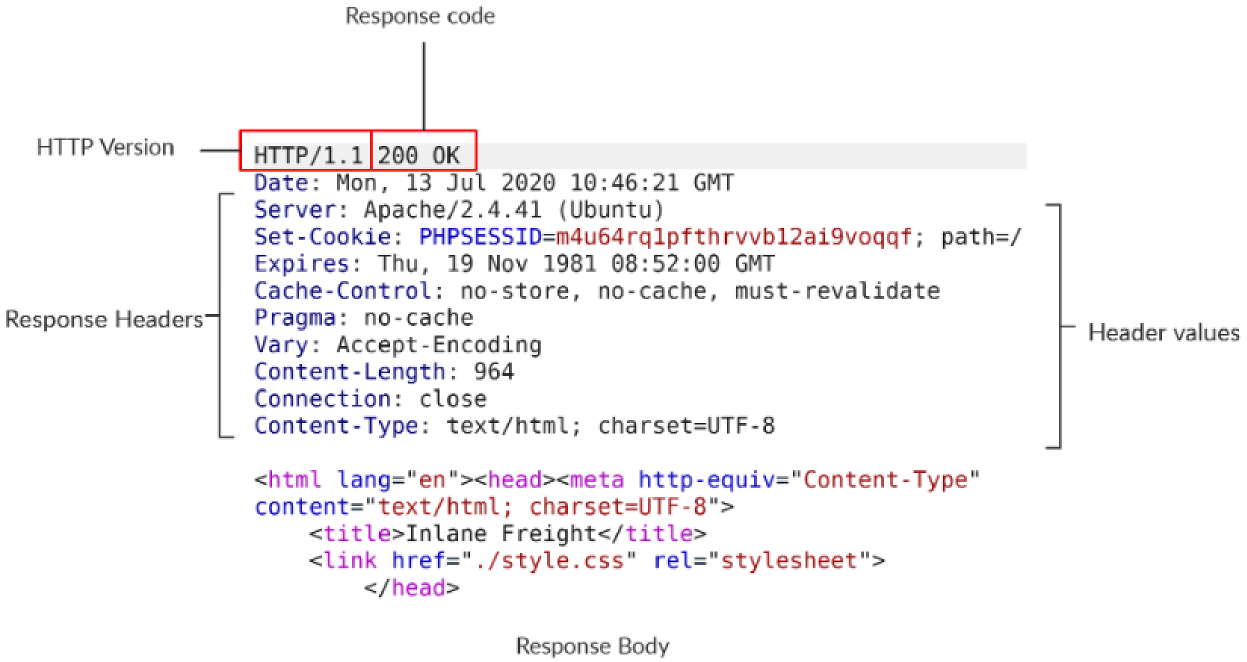
```
GET /users/login.html HTTP/1.1
Host: inlanefreight.com
User-Agent: Mozilla/5.0 (Ubuntu; Linux x86_64;) Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: text/html; charset=UTF-8
Connection: close
Cookie: PHPSESSID=c4ggt4jull9obt7aupa55o8vbf
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

Figure 14: HTTP GET request to the URL: ***http://inlanefreight.com/users/login.html***

Field	Example	Description
Method	GET	The HTTP method or verb, which specifies the type of action to perform.
Path	/users/login.html	The path to the resource being accessed. This field can also be suffixed with a query string (e.g. ?username=user).
Version	HTTP/1.1	The third and final field is used to denote the HTTP version.

The next set of lines contain HTTP header value pairs, like **Host, User-Agent, Cookie,** and many other possible headers. These headers are used to specify various attributes of a request. The headers are terminated with a new line, which is necessary for the server to validate the request. Finally, a request may end with the request body and data.

HTTP Response



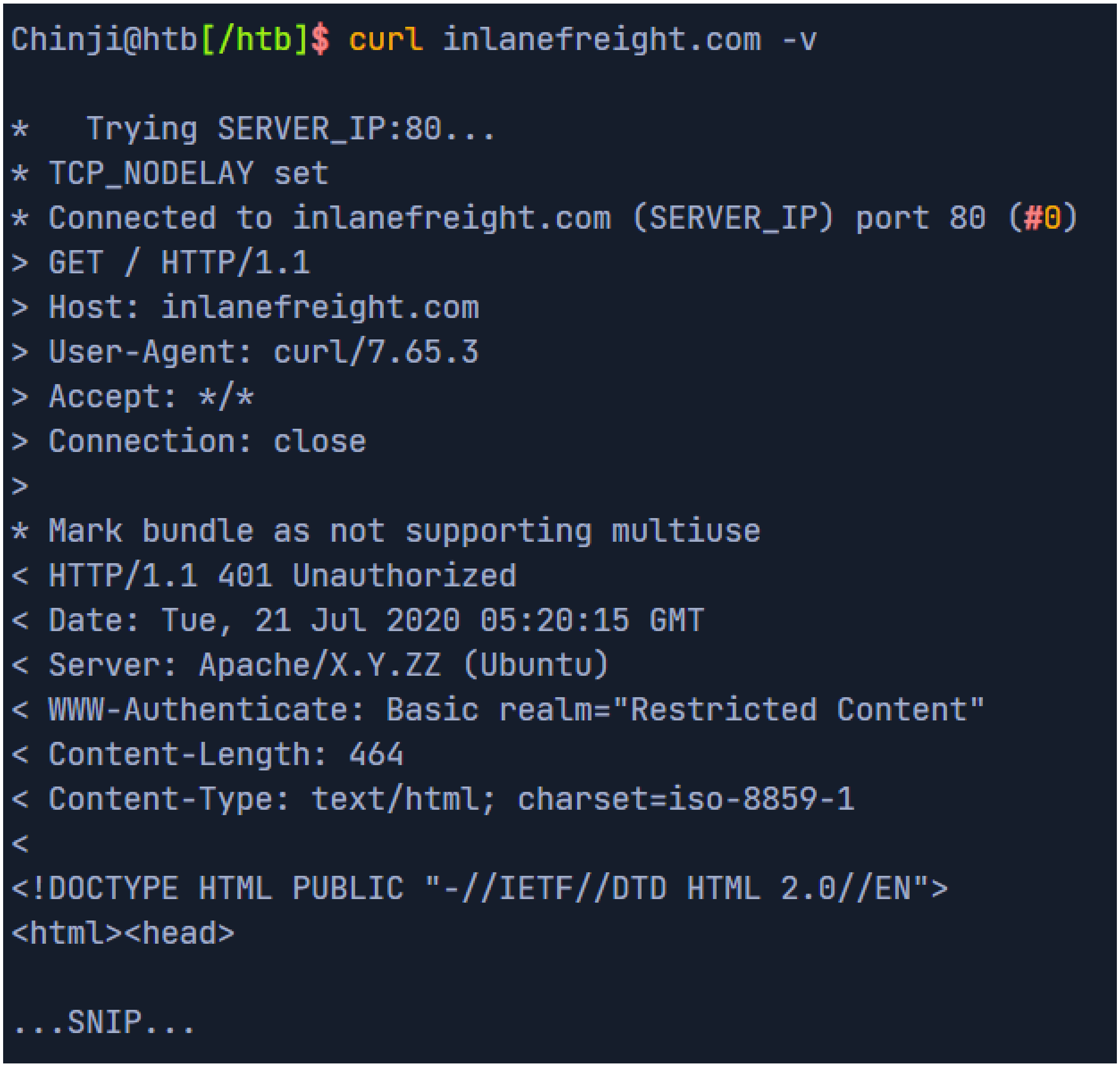
```
HTTP/1.1 200 OK
Date: Mon, 13 Jul 2020 10:46:21 GMT
Server: Apache/2.4.41 (Ubuntu)
Set-Cookie: PHPSESSID=m4u64rqlpfthrvvb12ai9voqqf; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 964
Connection: close
Content-Type: text/html; charset=UTF-8

<html lang="en"><head><meta http-equiv="Content-Type"
content="text/html; charset=UTF-8">
    <title>Inlane Freight</title>
    <link href="./style.css" rel="stylesheet">
        </head>
```

An HTTP response starts with a line containing the **HTTP version** (e.g., HTTP/1.1) followed by the **response code** (e.g., 200 OK), which indicates the status of the request.

After this, the response includes headers, much like an HTTP request. These headers provide additional information and will be covered in the next section. The response may also include a body, separated from the headers by a blank line. This body often contains HTML but can also include other content types like JSON, images, CSS, JavaScript, or files such as PDFs hosted on the server.

cURL

cURL provides the option to **view the complete HTTP request and response**, which is especially useful during web penetration testing or when developing exploits. By using the **-v (verbose) flag** in a cURL command, you can see **detailed output of both the request sent to the server and the response received.**



```
Chinji@htb[/htb]$ curl inlanefreight.com -v

*   Trying SERVER_IP:80...
* TCP_NODELAY set
* Connected to inlanefreight.com (SERVER_IP) port 80 (#0)
> GET / HTTP/1.1
> Host: inlanefreight.com
> User-Agent: curl/7.65.3
> Accept: */*
> Connection: close
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 401 Unauthorized
< Date: Tue, 21 Jul 2020 05:20:15 GMT
< Server: Apache/X.Y.ZZ (Ubuntu)
< WWW-Authenticate: Basic realm="Restricted Content"
< Content-Length: 464
< Content-Type: text/html; charset=iso-8859-1
<
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>

...SNIP...
```

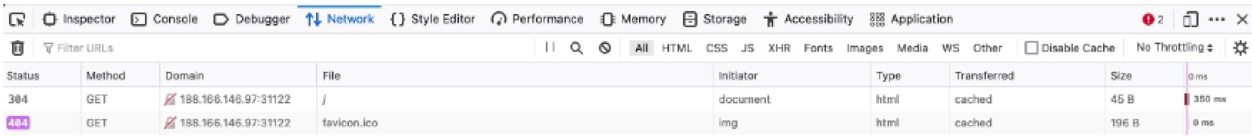
Figure 15: Full HTTP request and response

***Note**: The -vvv flag shows an even more verbose output*

Browser DevTools

Modern browsers like Chrome and Firefox include built-in Developer Tools (DevTools), originally designed to help developers test web applications. However, these tools are also valuable for web penetration testers, as they are readily available and provide deep insight into web activity during assessments. When accessing a website, the browser sends multiple HTTP requests and receives responses to build the visible page. By opening DevTools using **CTRL+SHIFT+I** or **F12**, testers can monitor this activity.

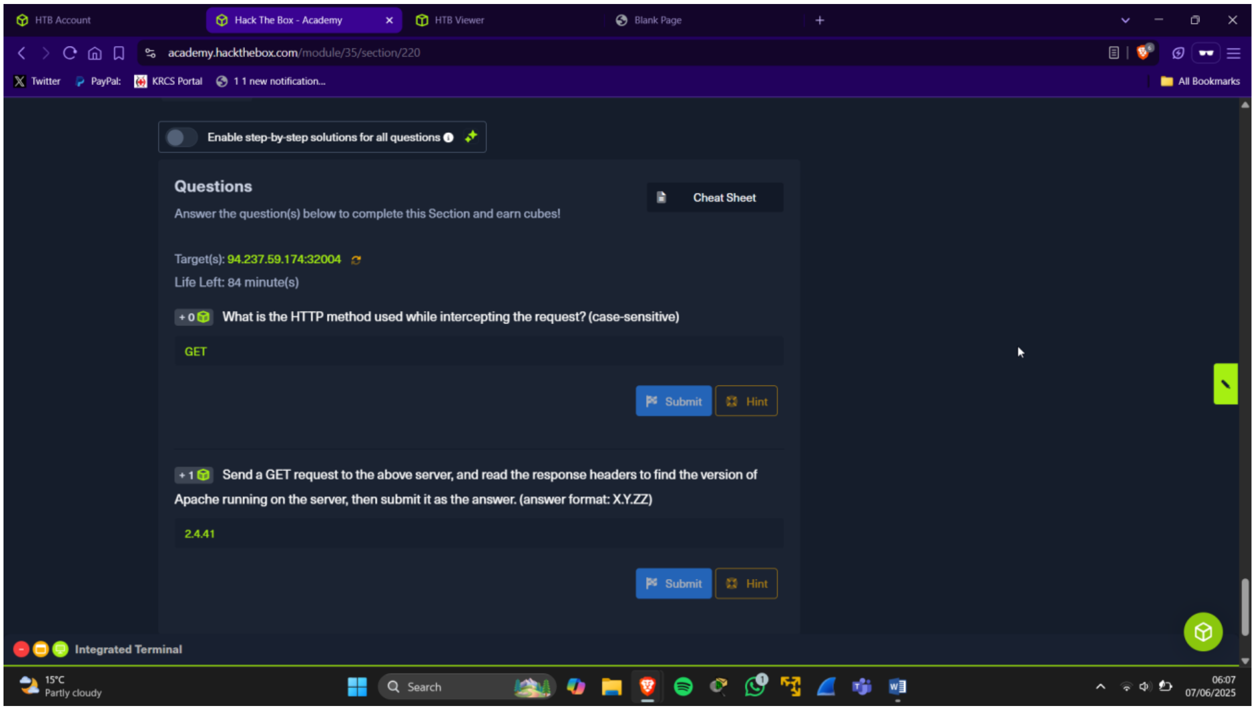
This module mainly focuses on the **Network tab**, which displays and tracks all web requests made by the browser.



Status	Method	Domain	File	Initiator	Type	Transferred	Size	0 ms
304	GET	188.166.146.97:31122	/	document	html	cached	45 B	350 ms
404	GET	188.166.146.97:31122	favicon.ico	img	html	cached	196 B	0 ms

Figure 16: DevTools show at a glance the ***response status, request method used, requested resource and the requested path***

Week Three: Assignment 2: HTB Academy - Web Requests



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): 94.237.59.174:32004

Life Left: 84 minute(s)

+0 What is the HTTP method used while intercepting the request? (case-sensitive)

GET

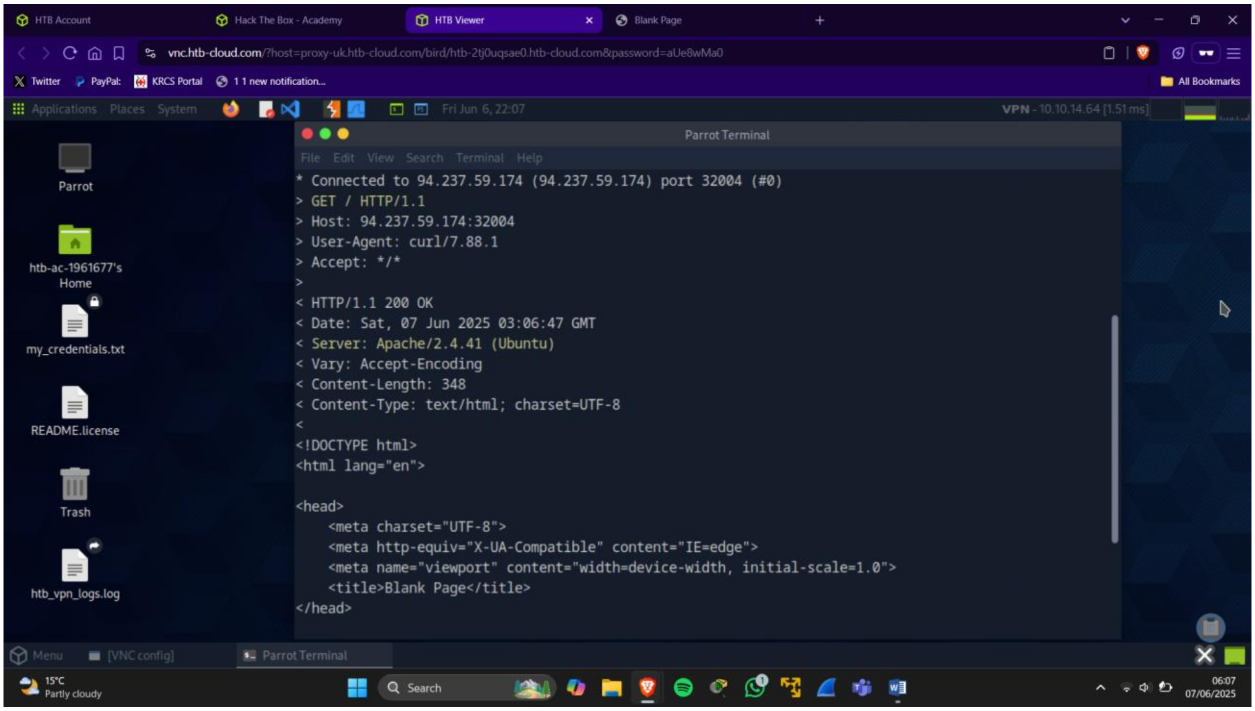
Submit Hint

+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

Figure 17: Answers to module questions



```
* Connected to 94.237.59.174 (94.237.59.174) port 32004 (#0)
> GET / HTTP/1.1
> Host: 94.237.59.174:32004
> User-Agent: curl/7.88.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Sat, 07 Jun 2025 03:06:47 GMT
< Server: Apache/2.4.41 (Ubuntu)
< Vary: Accept-Encoding
< Content-Length: 348
< Content-Type: text/html; charset=UTF-8
<
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Blank Page</title>
</head>
```

Figure 18: Evidence to the answers above

Please Note: I had to create another instance of pwnbox

HTTP Headers

HTTP headers are used to exchange information between the client and the server. Some headers are specific to requests, others to responses, while certain general headers apply to both. Each header is written with its name followed by a colon and its value(s), and multiple values can be included.

Headers are generally grouped into five categories:

- **General Headers**
- **Entity Headers**
- **Request Headers**
- **Response Headers**
- **Security Headers**

These headers help control how requests and responses are processed and interpreted.

General Headers

General headers are used in both HTTP requests and responses. They are contextual and are used to describe the message rather than its contents.

Header	Example	Description
Date	Date: Wed, 16 Feb 2022 10:38:44 GMT	Holds the date and time at which the message originated. It's preferred to convert the time to the standard UTC time zone.
Connection	Connection: close	Dictates if the current network connection should stay alive after the request finishes. Two commonly used values for this header are close and keep-alive. The close value from either the client or server means that they would like to terminate the connection, while the keep-alive header indicates that the connection should remain open to receive more data and input.

Entity Headers

These headers are used to describe the content (entity) transferred by a message. They are usually fo
und in responses and POST or PUT requests.

Header	Example	Description
Content-Type	Content-Type: text/html	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 charset field denotes the encoding standard, such as UTF-8.

Media-Type	Media-Type: application/pdf	The media-type is similar to Content-Type, and describes the data being transferred. This header can play a crucial role in making the server interpret our input. The charset field may also be used with this header.
Boundary	boundary="b4e4fbd93540"	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 --b4e4fbd93540 to separate different parts of the form.
Content-Length	Content-Length: 385	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.
Content-Encoding	Content-Encoding: gzip	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 Content-Encoding header.

Request Headers

These headers are used in an HTTP request and do not relate to the content of the message.

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 User-Agent 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 https://google.com the referer. Trusting this header can be dangerous as it can be easily manipulated, leading to unintended consequences.
Accept	Accept: */*	The Accept header describes which media types the client can understand. It can contain multiple media types separated by commas. The */* value signifies that all media types are accepted.
Cookie	Cookie: PHPSESSID=b4e4fbd93540	Contains cookie-value pairs in the format name=value. A cookie 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.

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 Cookie 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

HTTP Security headers are a class of response headers used to specify certain rules and policies to be followed by the browser while accessing the website.

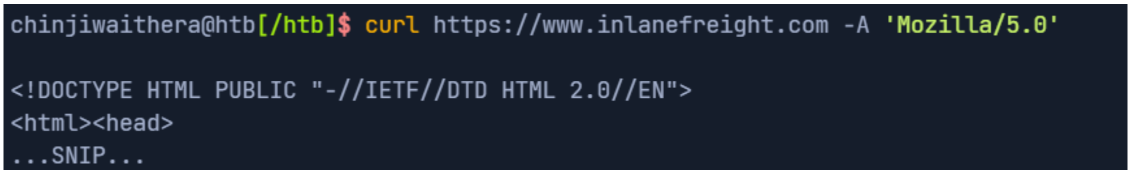
Header	Example	Description
Content-Security-Policy	Content-Security-Policy: script-src 'self'	Dictates the website's policy towards externally injected resources. This could be JavaScript code as well as script resources. This header instructs the browser to accept resources only from certain trusted domains, hence preventing attacks such as Cross-site scripting (XSS).
Strict-Transport-Security	Strict-Transport-Security: max-age=31536000	Prevents the browser from accessing the website over the plaintext HTTP protocol, and forces all communication to be carried over the secure HTTPS

		protocol. This prevents attackers from sniffing web traffic and accessing protected information such as passwords or other sensitive data.
Referrer-Policy	Referrer-Policy: origin	Dictates whether the browser should include the value specified via the Referer header or not. It can help in avoiding disclosing sensitive URLs and information while browsing the website.

cURL

Previously, we used the `-v` flag in cURL to view full HTTP requests and responses. If we only want to see the **response headers**, we can use the `-I` flag, which sends a **HEAD** request and returns just the headers. On the other hand, the `-i` flag includes both the **headers and the response body** (like HTML). The key difference is that `-I` changes the request type to HEAD, while `-i` keeps the original request type but adds headers to the output.

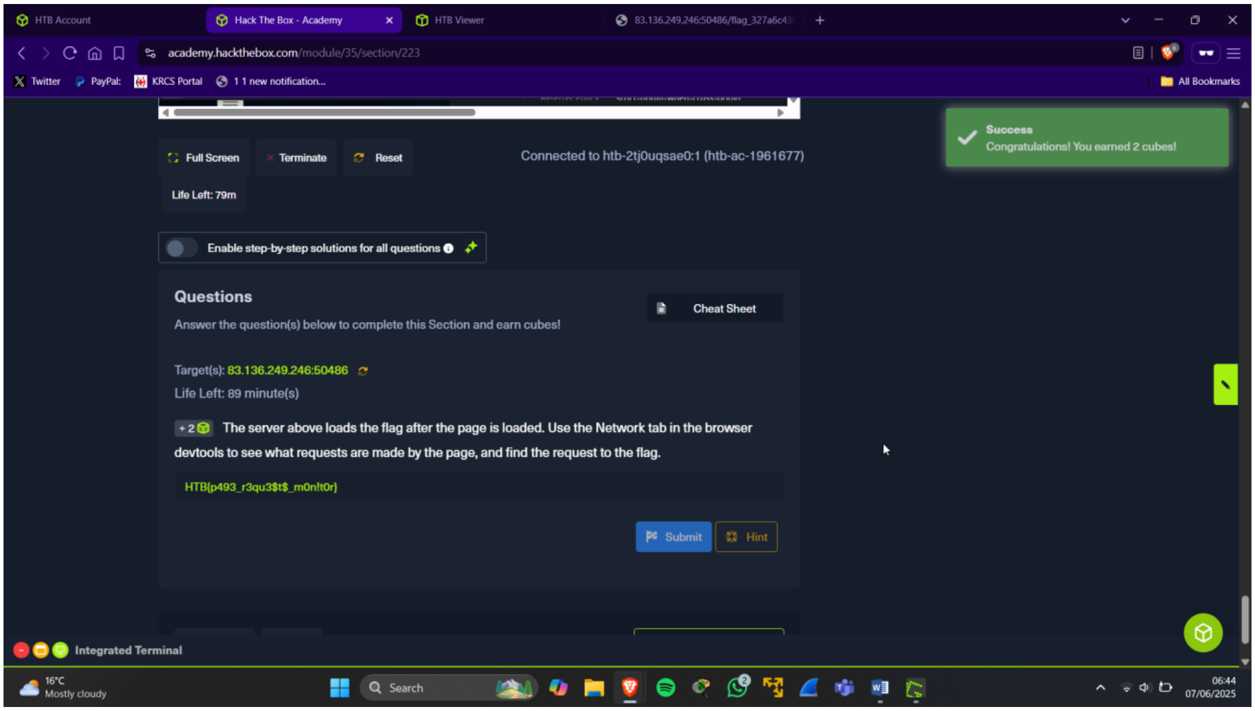
In addition to viewing headers, cURL also allows us to set request headers with the -**H flag**, as we will see in a later section. Some headers, like the User-Agent or Cookie headers, have their own flags. For example, we can use the -A to set our User-Agent, as follows:



```
chinjiwaithera@htb[/htb]$ curl https://www.inlanefreight.com -A 'Mozilla/5.0'

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

Figure 19: Fla –A to set the User Agent

Browser Dev Tools



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

Full Screen · Terminate · Reset

Connected to htb-2tj0uqsae0:1 (htb-ac-1961677)

Life Left: 79m

Success
Congratulations! You earned 2 cubes!

Enable step-by-step solutions for all questions

Questions

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

Target(s): 83.136.249.246:50486

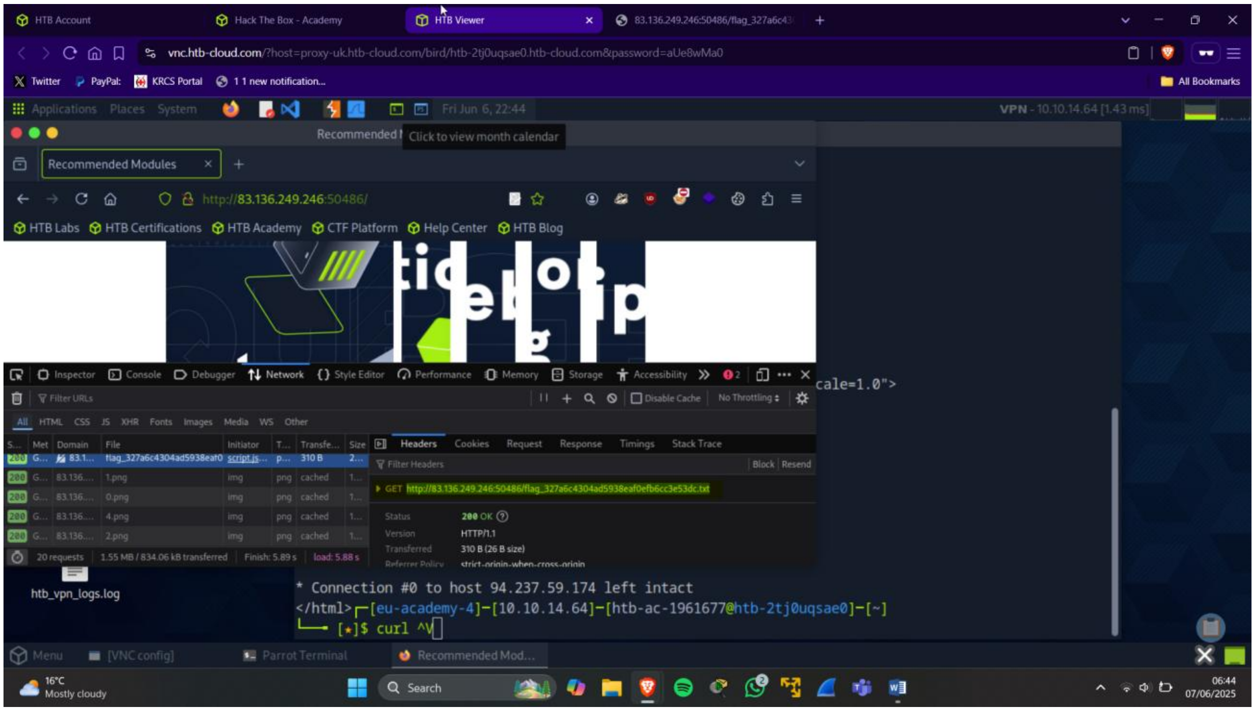
Life Left: 89 minute(s)

+ 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_m0n!t0r}

Submit · Hint

Figure 20: Answer to module question



vnc.htb-cloud.com/?host=proxy-uk.htb-cloud.com/bird/htb-2tj0uqsae0.htb-cloud.com&password=aUe8wMa0

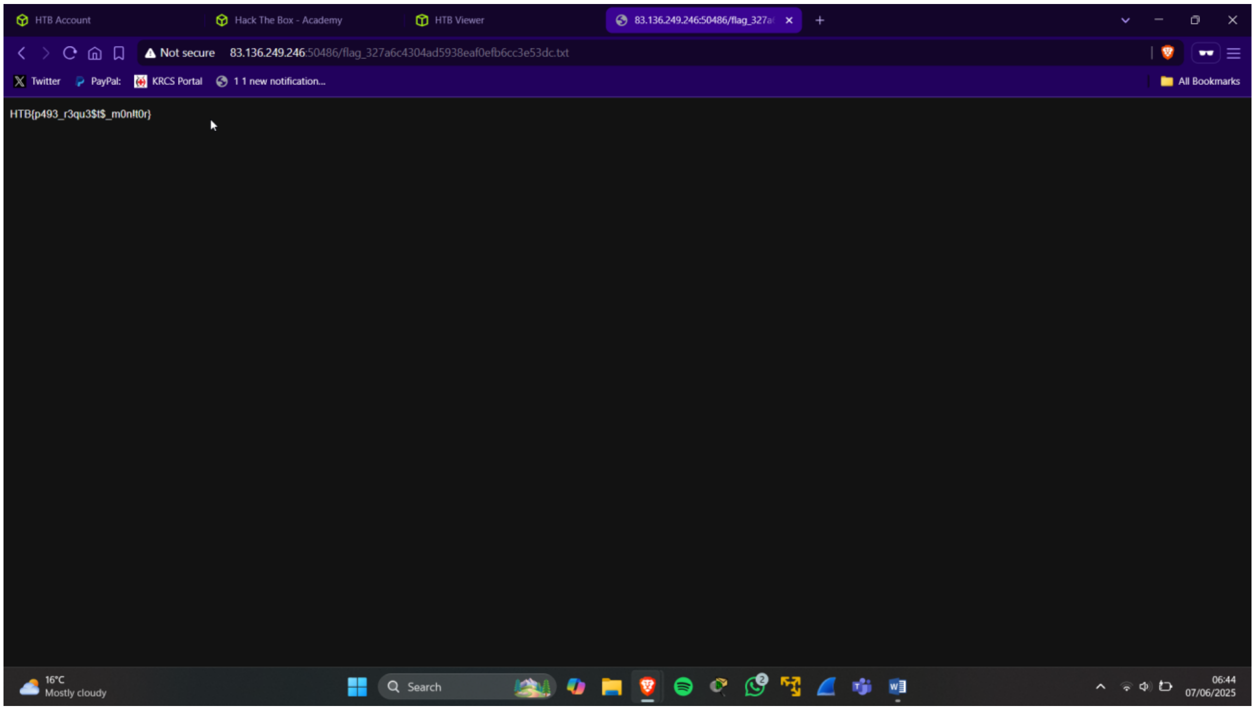
http://83.136.249.246:50486/

```
S...  Met  Domain     File                              Initiator   T...  Transfe...  Size
200   G... 83.1...    flag_327a6c4304ad5938eaf0         script.js... p...  310 B       2...
200   G... 83.136.... 1.png                             img         png   cached      1...
200   G... 83.136.... 0.png                             img         png   cached      1...
200   G... 83.136.... 4.png                             img         png   cached      1...
200   G... 83.136.... 2.png                             img         png   cached      1...
20 requests | 1.55 MB / 834.06 kB transferred | Finish: 5.89 s | load: 5.88 s
```

```
GET http://83.136.249.246:50486/flag_327a6c4304ad5938eaf0efb6cc3e53dc.txt
Status           200 OK
Version          HTTP/1.1
Transferred      310 B (26 B size)
Referrer Policy  strict-origin-when-cross-origin
```

```
* Connection #0 to host 94.237.59.174 left intact
</html>┌─[eu-academy-4]─[10.10.14.64]─[htb-ac-1961677@htb-2tj0uqsae0]─[~]
└──╼ [★]$ curl ^V
```

Figure 21: Evidence -1 of answer above



```
HTB{p493_r3qu3$t$_m0n!t0r}
```

Figure 22: Evidence -2 of answer above

HTTP Methods and Codes

HTTP provides various methods to interact with web resources. These methods tell the server what kind of action to perform with the request (like retrieving data, submitting forms, or uploading files). The method used appears at the start of an HTTP request, for example: `GET / HTTP/1.1`.

When using cURL with the `-v` flag, the method is visible in the first line of the request. In browser DevTools, it's shown in the **Method** column. The server's response includes an **HTTP response code**, indicating whether the request was processed successfully or if there was an issue.

Request Methods

Method	Description
GET	Requests a specific resource. Additional data can be passed to the server via query strings in the URL (e.g. ?param=value).
POST	Sends data to the server. It can handle multiple types of input, such as text, PDFs, and other forms of binary data. This data is appended in the request body present after the headers. The POST method is commonly used when sending information (e.g. forms/logins) or uploading data to a website, such as images or documents.
HEAD	Requests the headers that would be returned if a GET request was made to the server. It doesn't return the request body and is usually made to check the response length before downloading resources.
PUT	Creates new resources on the server. Allowing this method without proper controls can lead to uploading malicious resources.
DELETE	Deletes an existing resource on the webserver. If not properly secured, can lead to Denial of Service (DoS) by deleting critical files on the web server.
OPTIONS	Returns information about the server, such as the methods accepted by it.
PATCH	Applies partial modifications to the resource at the specified location.

Response Codes

Type	Description
1xx	Provides information and does not affect the processing of the request.
2xx	Returned when a request succeeds.
3xx	Returned when the server redirects the client.
4xx	Signifies improper requests from the client. For example, requesting a resource that doesn't exist or requesting a bad format.
5xx	Returned when there is some problem with the HTTP server itself.

The following are some of the commonly seen examples from each of the above HTTP method types:

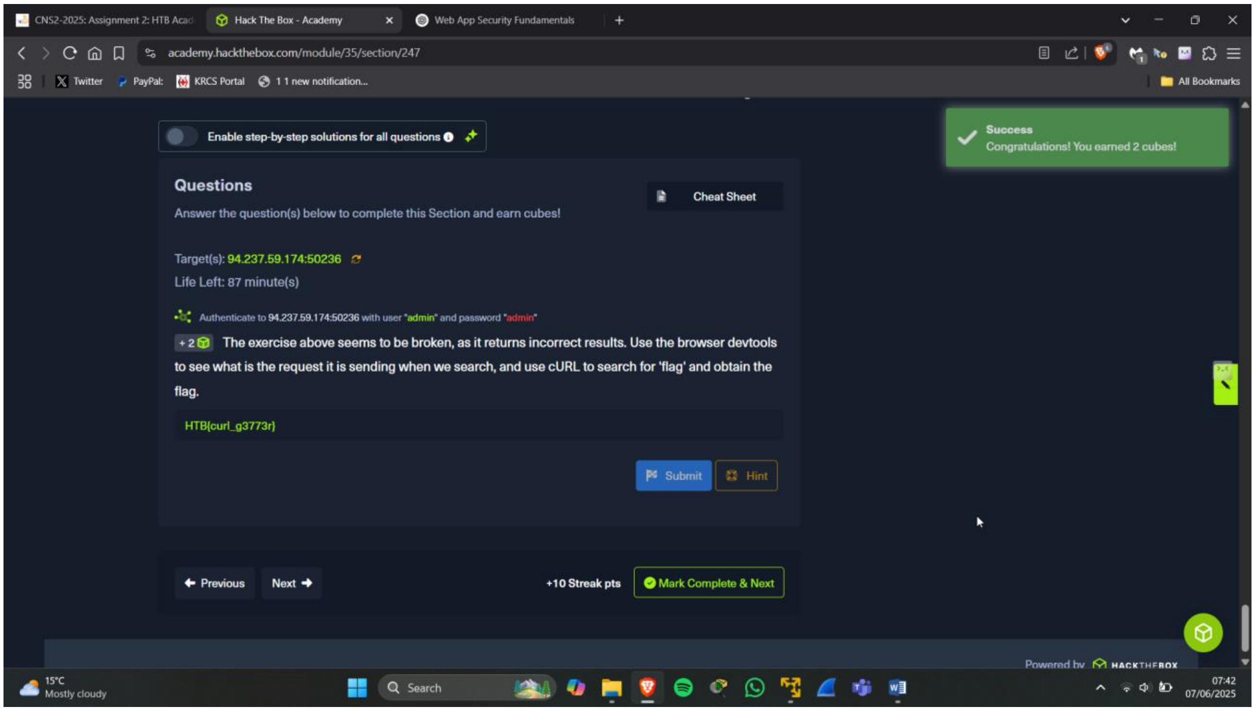
Code	Description
200 OK	Returned on a successful request, and the response body usually contains the requested resource.
302 Found	Redirects the client to another URL. For example, redirecting the user to their dashboard after a successful login.
400 Bad Request	Returned on encountering malformed requests such as requests with missing line terminators.

403 Forbidden	Signifies that the client doesn't have appropriate access to the resource. It can also be returned when the server detects malicious input from the user.
404 Not Found	Returned when the client requests a resource that doesn't exist on the server.
500 Internal Server Error	Returned when the server cannot process the request.

GET

Whenever we visit any URL, our browsers default to a GET request to obtain the remote resources hosted at that URL.

HTTP Basic Authorization



Success
Congratulations! You earned 2 cubes!

Questions

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

Target(s): 94.237.59.174:50236

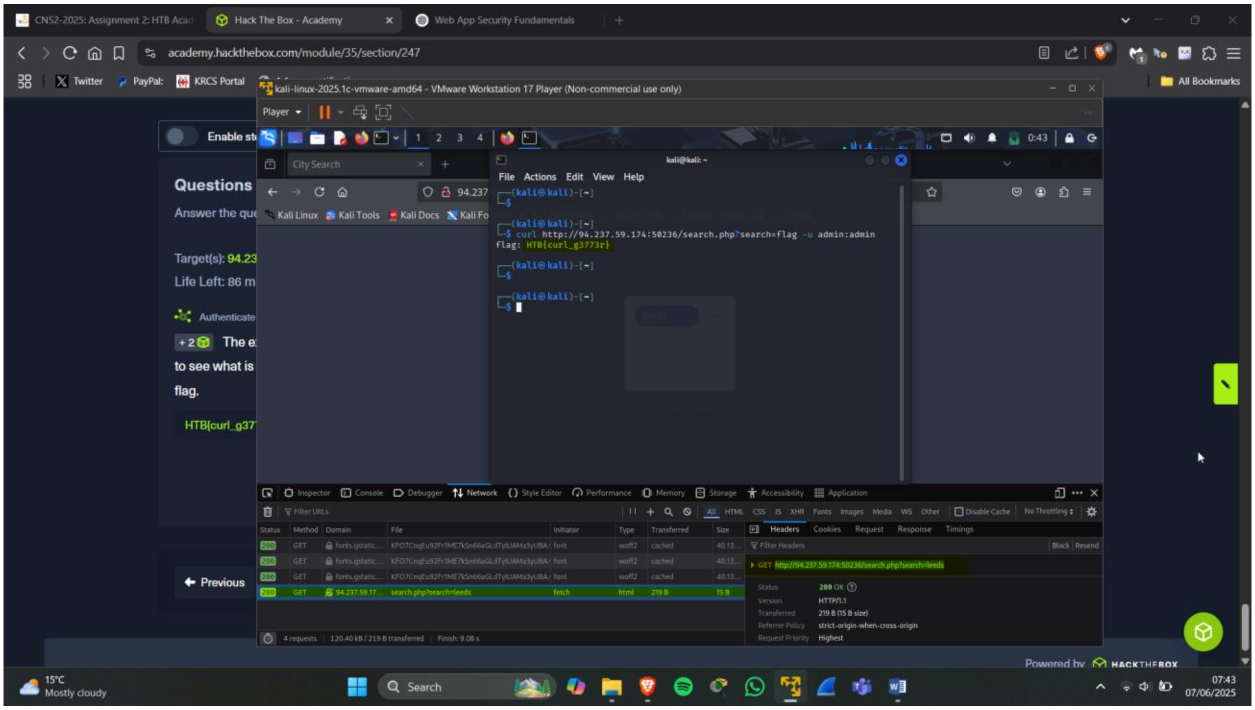
Life Left: 87 minute(s)

Authenticate to 94.237.59.174:50236 with user "admin" and password "admin"

+ 2 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}

Figure 23: Answer to the module question



```
┌──(kali㉿kali)-[~]
└─$ 

┌──(kali㉿kali)-[~]
└─$ curl http://94.237.59.174:50236/search.php?search=flag -u admin:admin
flag: HTB{curl_g3773r}

┌──(kali㉿kali)-[~]
└─$ 

┌──(kali㉿kali)-[~]
└─$ 
```

```
Status  Method  Domain              File                                          Initiator  Type   Transferred  Size
200     GET     fonts.gstatic....   KFO7CnqEu92Fr1ME7kSn66aGLdTylUAMa3yUBA...     font       woff2  cached       40.13...
200     GET     fonts.gstatic....   KFO7CnqEu92Fr1ME7kSn66aGLdTylUAMa3yUBA...     font       woff2  cached       40.13...
200     GET     fonts.gstatic....   KFO7CnqEu92Fr1ME7kSn66aGLdTylUAMa3yUBA...     font       woff2  cached       40.13...
200     GET     94.237.59.17...     search.php?search=leeds                       fetch      html   219 B        15 B

4 requests | 120.40 kB / 219 B transferred | Finish: 9.08 s
```

```
GET http://94.237.59.174:50236/search.php?search=leeds
Status           200 OK
Version          HTTP/1.1
Transferred      219 B (15 B size)
Referrer Policy  strict-origin-when-cross-origin
Request Priority Highest
```

Figure 24: Evidence to the answer above with VMware and not an instance

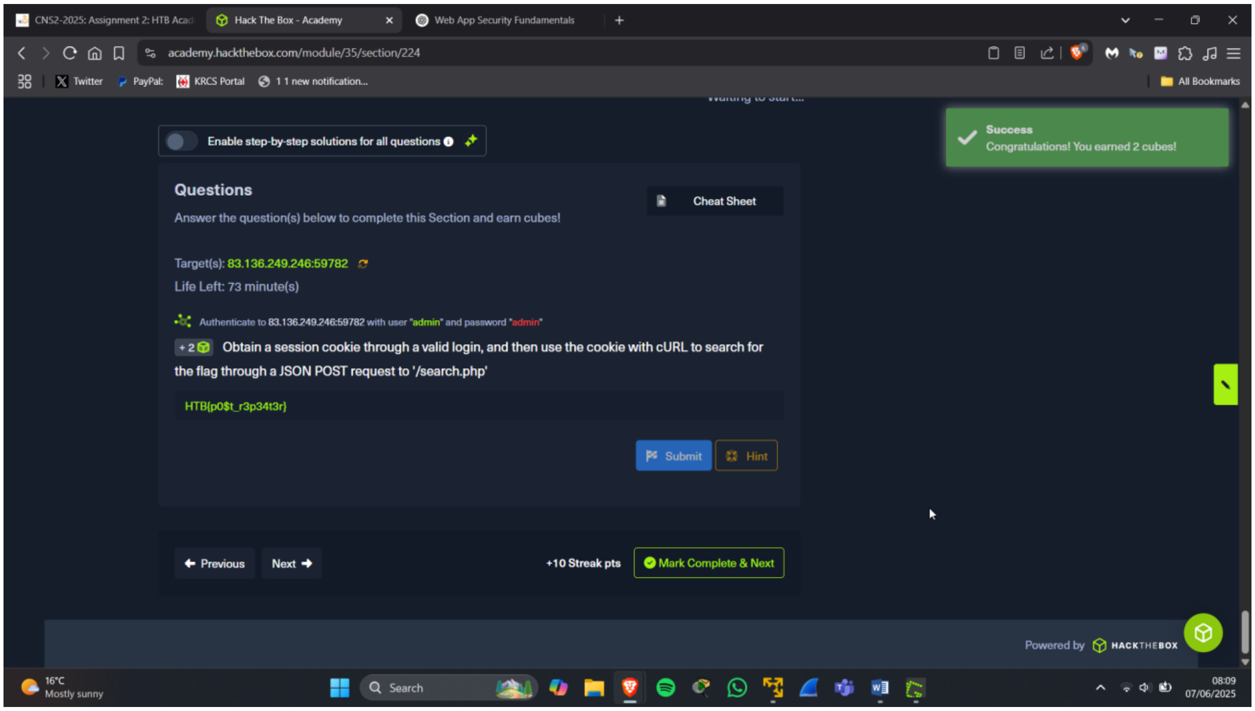
POST

While GET requests are often used to access pages or perform searches, POST requests are used when web applications need to transfer files or move user data out of the URL.

Unlike GET, which sends data in the URL, **POST** places data in the request body, offering several advantages:

- No URL logging of data: This is especially useful for large file uploads, which would otherwise clog server logs if done via GET.
- Fewer encoding limits: POST handles binary data better since it's not restricted to URL-safe characters.
- Larger data transfer: URLs have character limits (generally <2000), which makes GET unsuitable for sending large volumes of data, while POST can handle much more through the body.

Login Forms and Authenticated Cookies



Hack The Box - Academy
Web App Security Fundamentals
academy.hackthebox.com/module/35/section/224
Enable step-by-step solutions for all questions
Cheat Sheet
Submit
Hint
Previous
Next
+10 Streak pts
Mark Complete & Next
Powered by HACKTHEBOX

Success
Congratulations! You earned 2 cubes!

Questions

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

Target(s): 83.136.249.246:59782

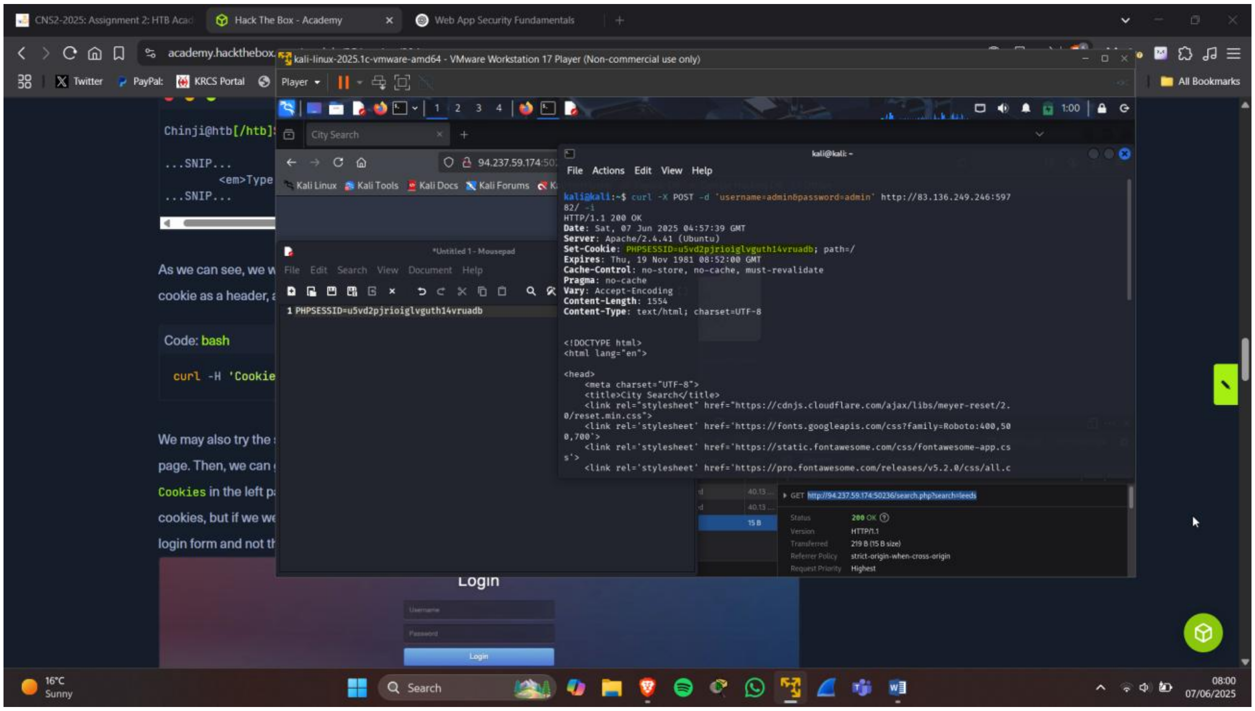
Life Left: 73 minute(s)

Authenticate to 83.136.249.246:59782 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}

Figure 25: Answer to the module question



kali-linux-2025.1c-vmware-amd64 - VMware Workstation 17 Player (Non-commercial use only)
City Search
*Untitled 1 - Mousepad
File Actions Edit View Help
kali@kali: ~
Login
Username
Password
Login

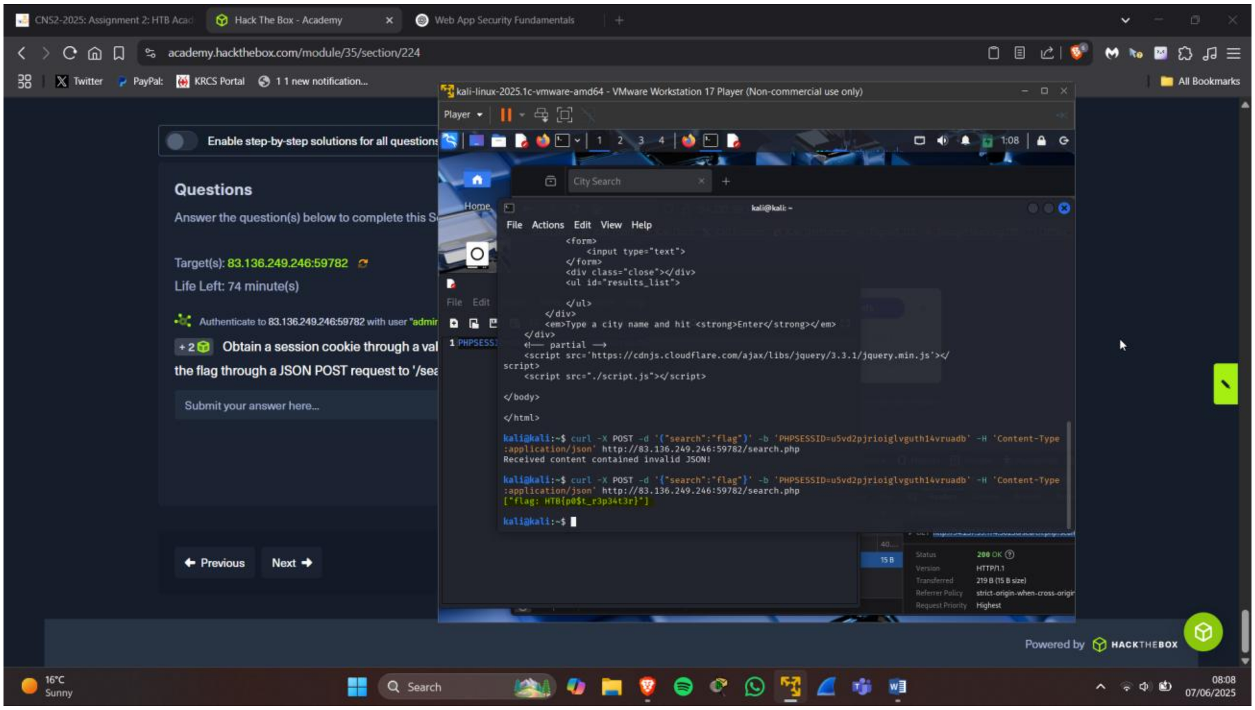
```
1 PHPSESSID=u5vd2pjrioiglvguth14vruadb
```

```
kali@kali:~$ curl -X POST -d 'username=admin&password=admin' http://83.136.249.246:597
82/ -i
HTTP/1.1 200 OK
Date: Sat, 07 Jun 2025 04:57:39 GMT
Server: Apache/2.4.41 (Ubuntu)
Set-Cookie: PHPSESSID=u5vd2pjrioiglvguth14vruadb; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 1554
Content-Type: text/html; charset=UTF-8


<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>City Search</title>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/meyer-reset/2.
0/reset.min.css">
    <link rel='stylesheet' href='https://fonts.googleapis.com/css?family=Roboto:400,50
0,700'>
    <link rel='stylesheet' href='https://static.fontawesome.com/css/fontawesome-app.cs
s'>
    <link rel='stylesheet' href='https://pro.fontawesome.com/releases/v5.2.0/css/all.c
```

GET http://94.237.59.174:50236/search.php?search=leeds

Status 200 OK
Version HTTP/1.1
Transferred 219 B (15 B size)
Referrer Policy strict-origin-when-cross-origin
Request Priority Highest

Figure 26: Getting the session cookie



academy.hackthebox.com/module/35/section/224

Questions

Answer the question(s) below to complete this S

Target(s): 83.136.249.246:59782

Life Left: 74 minute(s)

Authenticate to 83.136.249.246:59782 with user "admin

+2 Obtain a session cookie through a val the flag through a JSON POST request to '/sea

Submit your answer here...

```
            <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>
    <!-- partial -->
    <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=u5vd2pjrioiglvguth14vruadb' -H 'Content-Type
:application/json' http://83.136.249.246:59782/search.php
Received content contained invalid JSON!

kali@kali:~$ curl -X POST -d '{"search":"flag"}' -b 'PHPSESSID=u5vd2pjrioiglvguth14vruadb' -H 'Content-Type
:application/json' http://83.136.249.246:59782/search.php
["flag: HTB{p0$t_r3p34t3r}"]

kali@kali:~$ 
```

Status 200 OK
Version HTTP/1.1
Transferred 219 B (15 B size)
Referrer Policy strict-origin-when-cross-origi
Request Priority Highest

Hack The Box - Academy
Web App Security Fundamentals
Enable step-by-step solutions for all questions
kali-linux-2025.1c-vmware-amd64 - VMware Workstation 17 Player (Non-commercial use only)
City Search
kali@kali: ~
File Actions Edit View Help
Previous
Next
Powered by HACKTHEBOX
16°C Sunny
08:08
07/06/2025

Figure 27: Getting the flag with the session cookie

CRUD API

APIs

APIs come in various types, and many are designed to interact with databases. Through an API, we can specify the **table** and **row** we want to access or modify using a specific **HTTP method**.

CRUD

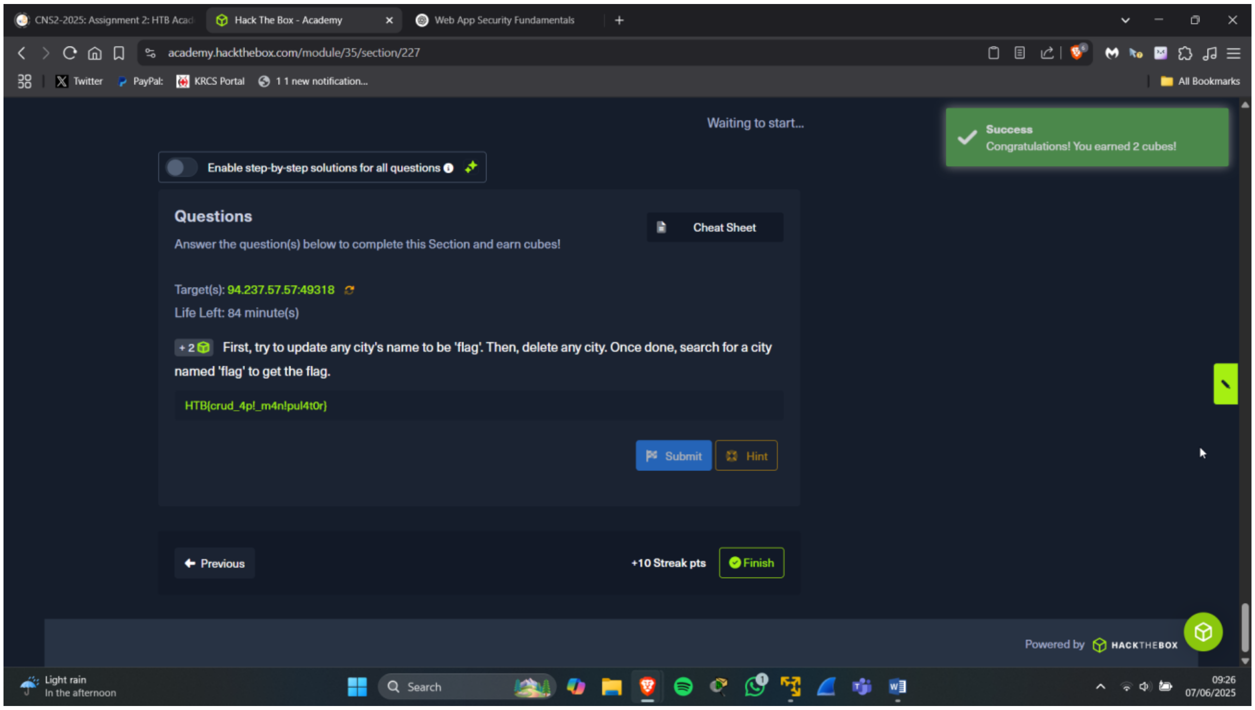
In general, APIs perform 4 main operations on the requested database entity:

Operation	HTTP Method	Description
Create	POST	Adds the specified data to the database table
Read	GET	Reads the specified entity from the database table
Update	PUT	Updates the data of the specified database table
Delete	DELETE	Removes the specified row from the database table

The four main operations—Create, Read, Update, and Delete (CRUD)—are commonly used in CRUD APIs, and also apply to REST APIs and other API types.

However, not all APIs function the same way. The actions we can perform and the data we can access depend on user permissions and access control set by the API.

Week Three: Assignment 2: HTB Academy - Web Requests



academy.hackthebox.com/module/35/section/227

Success
Congratulations! You earned 2 cubes!

Enable step-by-step solutions for all questions

Questions

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

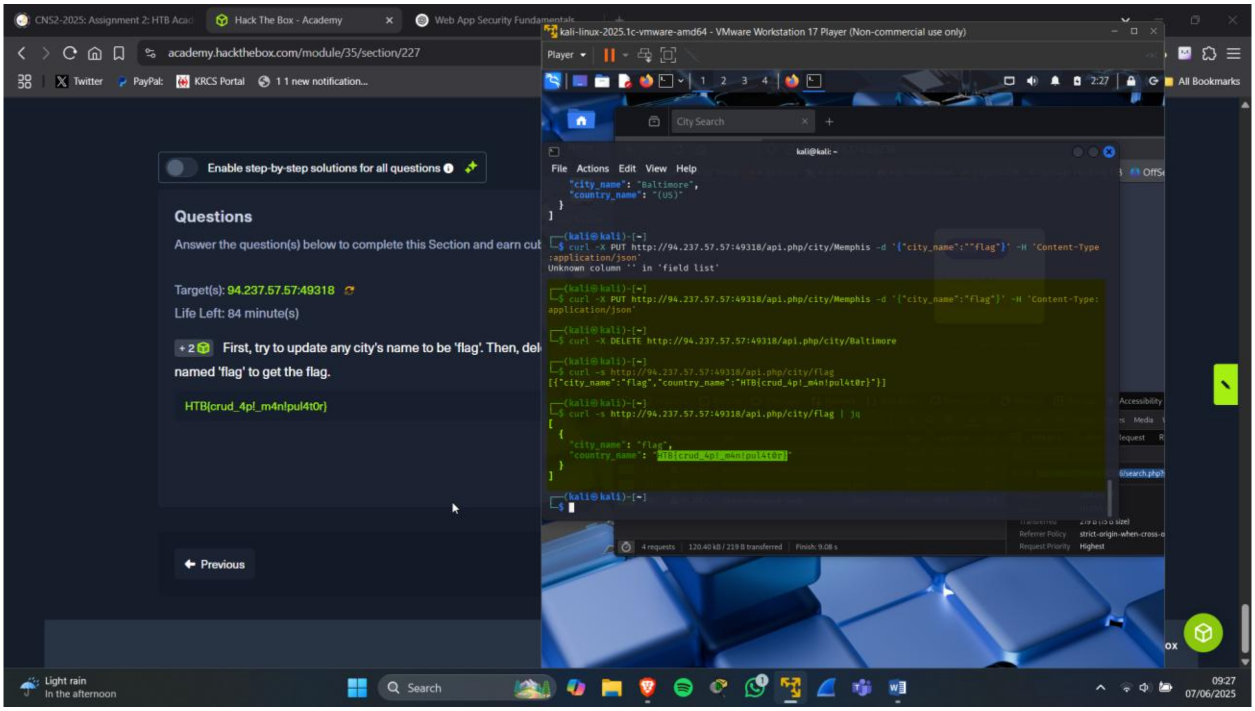
Target(s): 94.237.57.57:49318

Life Left: 84 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_4p!_m4n!pul4t0r}

Figure 28: Answer to the module question



```
    "city_name": "Baltimore",
    "country_name": "(US)"
  }
]

┌──(kali㉿kali)-[~]
└─$ curl -X PUT http://94.237.57.57:49318/api.php/city/Memphis -d '{"city_name":""flag"}' -H 'Content-Type
:application/json'
Unknown column '' in 'field list'

┌──(kali㉿kali)-[~]
└─$ curl -X PUT http://94.237.57.57:49318/api.php/city/Memphis -d '{"city_name":"flag"}' -H 'Content-Type:
application/json'

┌──(kali㉿kali)-[~]
└─$ curl -X DELETE http://94.237.57.57:49318/api.php/city/Baltimore

┌──(kali㉿kali)-[~]
└─$ curl -s http://94.237.57.57:49318/api.php/city/flag
[{"city_name":"flag","country_name":"HTB{crud_4p!_m4n!pul4t0r}"}]

┌──(kali㉿kali)-[~]
└─$ curl -s http://94.237.57.57:49318/api.php/city/flag | jq
[
  {
    "city_name": "flag",
    "country_name": "HTB{crud_4p!_m4n!pul4t0r}"
  }
]

┌──(kali㉿kali)-[~]
└─$ 
```

kali-linux-2025.1c-vmware-amd64 - VMware Workstation 17 Player (Non-commercial use only)
City Search
kali@kali: ~
File Actions Edit View Help
4 requests
120.40 kB / 219 B transferred
Finish: 9.08 s

Figure 29: Explanation to answer above, step by step

Conclusion

I’ve gained a solid foundation in how web communication truly works—starting from how a browser resolves a domain through DNS, to how requests and responses are structured and processed using HTTP and HTTPS. I've learned how tools like **cURL** and **Browser DevTools** help inspect, test, and manipulate these requests—skills that are essential in any web penetration testing workflow.

Understanding the difference between **GET and POST**, how **HTTPS encryption and handshakes** work, and how **APIs interact with databases** using CRUD operations, has deepened my grasp of real-world web interactions. I now see the significance of headers, methods, status codes, and how requests are routed securely or redirected when needed.

The module's hands-on exercises, particularly using cURL with flags like `-v`, `-i`, and `-I`, helped me visualize how data moves across the web. DevTools further complemented this by showing me how browsers manage and render web content through multiple layered requests.

This experience has not only strengthened my technical foundation, but also prepared me for CREST certification paths and future penetration testing challenges. Most importantly, I now understand that mastering the basics of HTTP/HTTPS and web request analysis isn't just a step—it's the *core* of everything I’ll build on in information security.