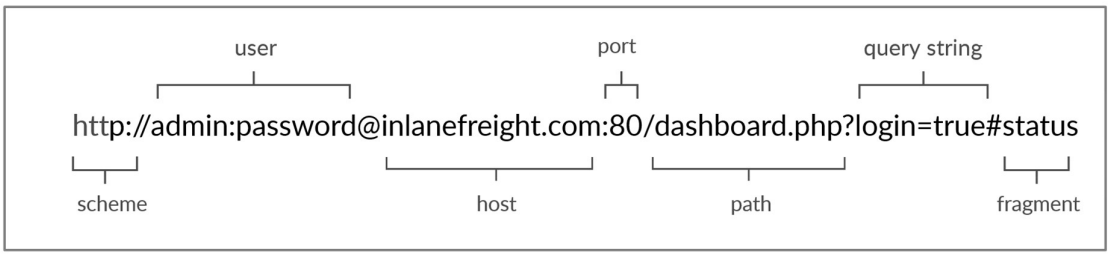
HyperText Transfer Protocol (HTTP)

HTTP is an application-level protocol used to access the World Wide Web resources. The term hypertext stands for text containing links to other resources and text that the readers can easily interpret

The default port for HTTP communication is **port 80**

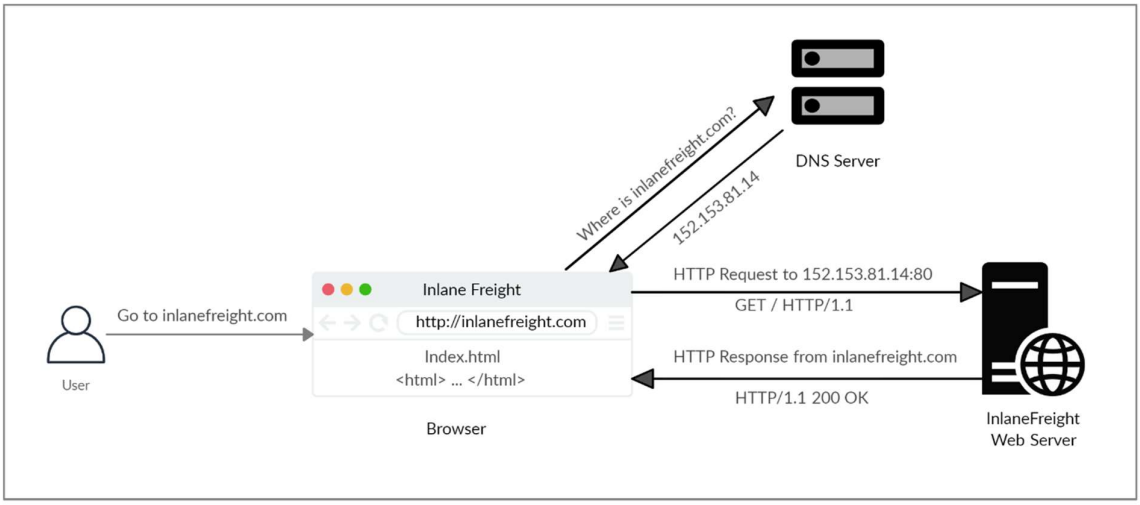
We enter a Fully Qualified Domain Name (FQDN) as a Uniform Resource Locator (URL) to reach the desired website, like https://sashakyotoz_dev

URL structure:



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:



cURL (client URL) is a command-line tool and library that primarily supports HTTP along with many other protocols.

HyperText Transfer Protocol (HTTP)

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

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

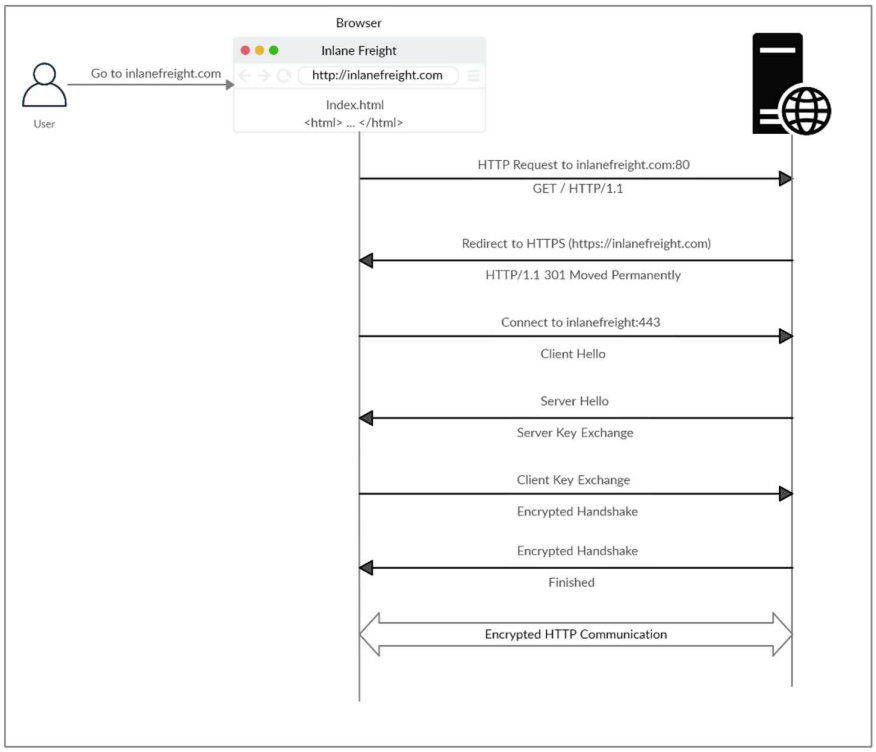
To download a page or a file and output the content into a file using the -O flag. If we want to specify the output file name, we can use the -o flag and specify the name.

HyperText Transfer Protocol (HTTP)

```
SashaKYotoz@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.
```

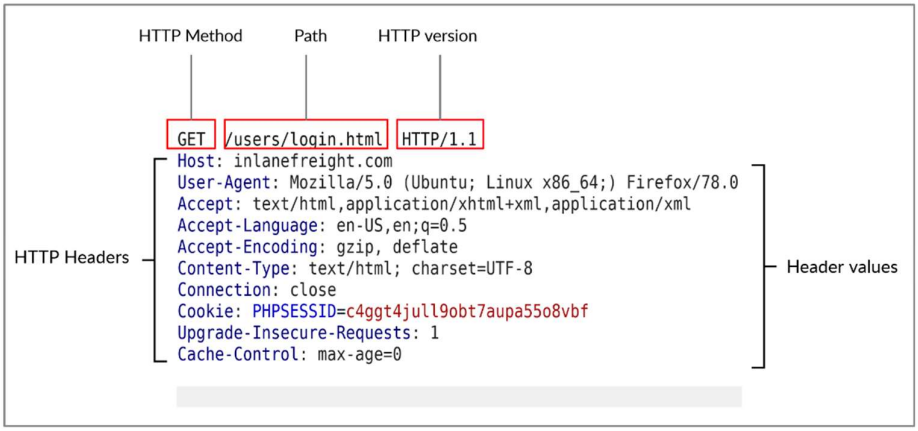
HTTPS FLOW:



If we type http:// instead of https:// to visit a website that enforces HTTPS, the browser attempts to resolve the domain and redirects the user to the webserver hosting the target website. A request is sent to port 80 first, server detects this and redirects the client to secure HTTPS port 443 instead. This is done via the 301 Moved Permanently response code

Client sends a "client hello" packet, giving information about itself. After this, the server replies with "server hello", followed by a key exchange to exchange SSL certificates

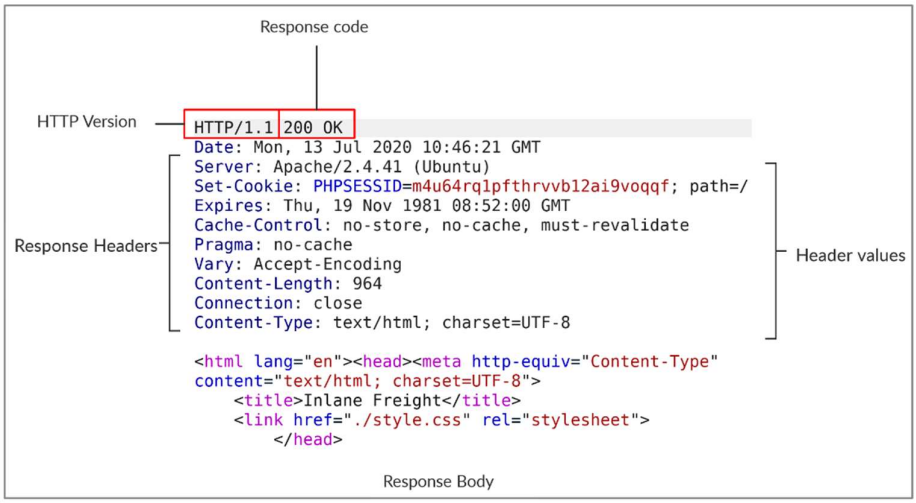
HTTP Request:



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.

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

HTTP Response:



The first line of an HTTP response contains two fields separated by spaces. The first being the HTTP version (e.g. HTTP/1.1), and the second denotes the HTTP response code (e.g. 200 OK).
Response codes are used to determine the request's status, as will be discussed in a later section. After the first line, the response lists its headers, similar to an HTTP request.

To view the full HTTP request and response, we can simply add the **-v** verbose flag to our earlier commands, and it should print both the request and response

HTTP Headers:

Headers can have one or multiple values, appended after the header name and separated by a colon. We can divide headers into the following categories:

1. General Headers
2. Entity Headers
3. Request Headers
4. Response Headers
5. Security 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 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

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.

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

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 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

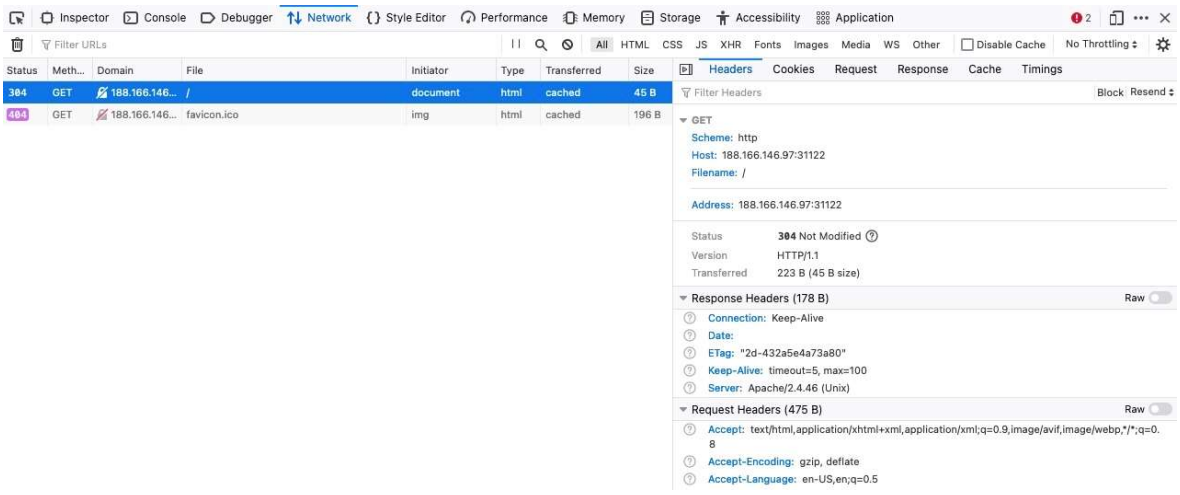
Header	Example	Description
`Server`	`Server: Apache/2.2.14 (Win32)`	Contains information about the HTTP server, which processed the request. It can be used to gain information about the server, such as its version, and enumerate it further.
`Set-Cookie`	`Set-Cookie: PHPSESSID=b4e4fbd93540`	Contains the 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.

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.

Browser DevTools:

We can go to the Network tab to view the different requests made by the page. We can click on any of the requests to view its details:



In the first Headers tab, we see both the HTTP request and HTTP response headers. The devtools automatically arrange the headers into sections, but we can click on the Raw button to view their details in their raw format. Furthermore, we can check the Cookies tab to see any cookies used by the request

HTTP Methods and Codes:

HTTP supports multiple methods for accessing a resource. In the HTTP protocol, several request methods allow the browser to send information, forms, or files to the server. These methods are used, among other things, to tell the server how to process the request we send and how to reply

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.

Status codes are used to tell the client the status of their request. An HTTP server can return five classes of status codes:

Class	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.

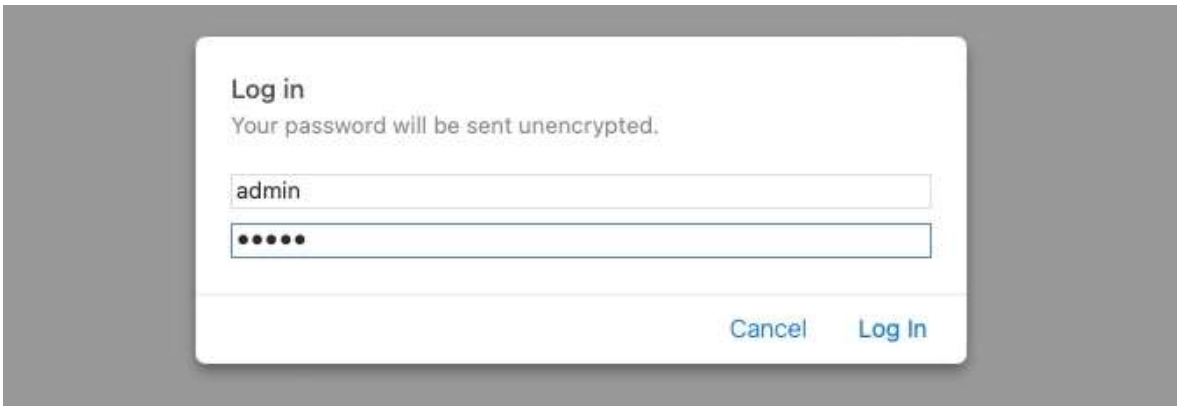
GET:

Whenever we visit any URL, our browsers default to a GET request to obtain the remote resources hosted at that URL. Once the browser receives the initial page it is requesting; it may send other requests using various HTTP methods

HTTP Basic Auth

Website prompts us to enter a username and a password. Unlike the usual login forms, which utilize HTTP parameters to validate the user credentials (e.g. POST request), this type of authentication utilizes a basic HTTP authentication, which is handled directly by the webserver to protect a specific page/directory, without directly interacting with the web application

To access the page, we have to enter a valid pair of credentials, which are admin:admin in this case:



There is another method we can provide the basic HTTP auth credentials, which is directly through the URL as (username:password@URL)

GET

```
SashaKYotoz@htb[/htb]$ curl http://admin:admin@<SERVER_IP>:<PORT>/

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

<head>
...SNIP...
```

HTTP Authorization Header

As we are using basic HTTP auth, we see that our HTTP request sets the Authorization header to Basic YWRtaW46YWRtaW4=, which is the base64 encoded value of admin:admin. If we were using a modern method of authentication (e.g. JWT), the Authorization would be of type Bearer and would contain a longer encrypted token

We can set the header with the -H flag, and will use the same value from the above HTTP request. We can add the -H flag multiple times to specify multiple headers:

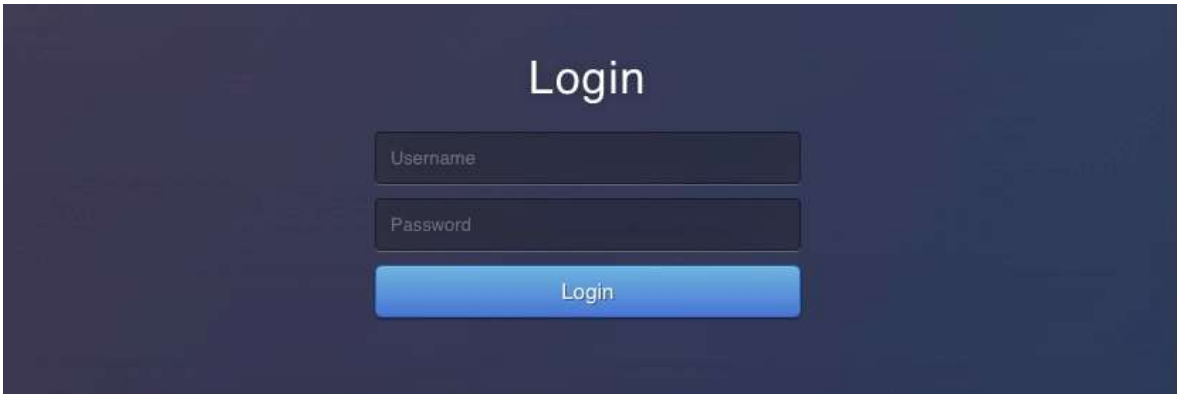
POST:

Whenever web applications need to transfer files or move the user parameters from the URL, they utilize POST requests

Unlike HTTP `GET`, which places user parameters within the URL, HTTP `POST` places user parameters within the HTTP Request body. This has three main benefits:

- `Lack of Logging`: As POST requests may transfer large files (e.g. file upload), it would not be efficient for the server to log all uploaded files as part of the requested URL, as would be the case with a file uploaded through a GET request.
- `Less Encoding Requirements`: URLs are designed to be shared, which means they need to conform to characters that can be converted to letters. The POST request places data in the body which can accept binary data. The only characters that need to be encoded are those that are used to separate parameters.
- `More data can be sent`: The maximum URL Length varies between browsers (Chrome/Firefox/IE), web servers (IIS, Apache, nginx), Content Delivery Networks (Fastly, Cloudfront, Cloudflare), and even URL Shorteners (bit.ly, amzn.to). Generally speaking, a URL's lengths should be kept to below 2,000 characters, and so they cannot handle a lot of data.

Login Forms



We will use the -X POST flag to send a POST request. Then, to add our POST data, we can use the -d flag and add the above data after it, as follows:

```
POST

SashaKYotoz@htb[/htb]$ curl -X POST -d 'username=admin&password=admin' http://<SERVER_IP>:<F

...SNIP...
        <em>Type a city name and hit <strong>Enter</strong></em>
...SNIP...
```

If we examine the HTML code, we will not see the login form code, but will see the search function code, which indicates that we did indeed get authenticated.

Authenticated Cookies

If we were successfully authenticated, we should have received a cookie so our browsers can persist our authentication, and we don't need to login every time we visit the page. We can use the -v or -i flags to view the response, which should contain the Set-Cookie header with our authenticated cookie:

```
POST

SashaKYotoz@htb[/htb]$ curl -X POST -d 'username=admin&password=admin' http://<SERVER_IP>:<

HTTP/1.1 200 OK
Date:
Server: Apache/2.4.41 (Ubuntu)
Set-Cookie: PHPSESSID=c1nsa6op7vtk7kdis7bcnbadf1; path=/

...SNIP...
        <em>Type a city name and hit <strong>Enter</strong></em>
...SNIP...
```

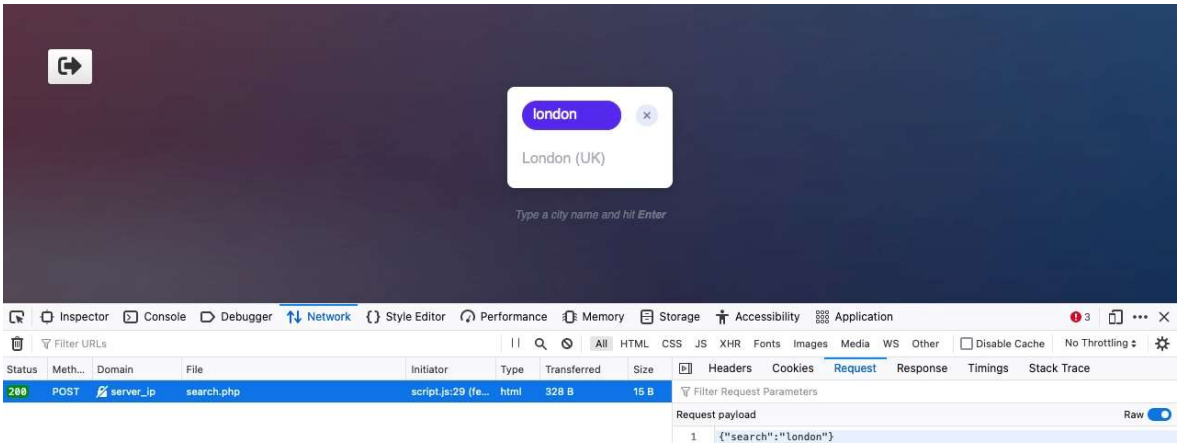
With our authenticated cookie, we should now be able to interact with the web application without needing to provide our credentials every time. To test this, we can set the above cookie with the -b flag in cURL, as follows:

```
POST

SashaKYotoz@htb[/htb]$ curl -b 'PHPSESSID=c1nsa6op7vtk7kdis7bcnbadf1' http://<SERVER_IP>:<PO

...SNIP...
        <em>Type a city name and hit <strong>Enter</strong></em>
...SNIP...
```

JSON Data



The POST data appear to be in JSON format, so our request must have specified the Content-Type header to be application/json. We can confirm this by right-clicking on the request, and selecting Copy>Copy Request Headers:

Code: bash

```
POST /search.php HTTP/1.1
Host: server_ip
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:97.0) Gecko/20100101 Firefox/97
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://server_ip/index.php
Content-Type: application/json
Origin: http://server_ip
Content-Length: 19
DNT: 1
Connection: keep-alive
Cookie: PHPSESSID=c1nsa6op7vtk7kdis7bcnbadf1
```

CRUD API:

APIs are used to interact with a database, such that we would be able to specify the requested table and the requested row within our API query, and then use an HTTP method to perform the operation needed. For example, for the api.php endpoint in our example, if we wanted to update the city table in the database, and the row we will be updating has a city name of london, then the URL would look something like this:

Code: bash

```
curl -X PUT http://<SERVER_IP>:<PORT>/api.php/city/london ...SNIP...
```

CRUD

We may utilize different HTTP methods to perform different operations on that row. 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

Read

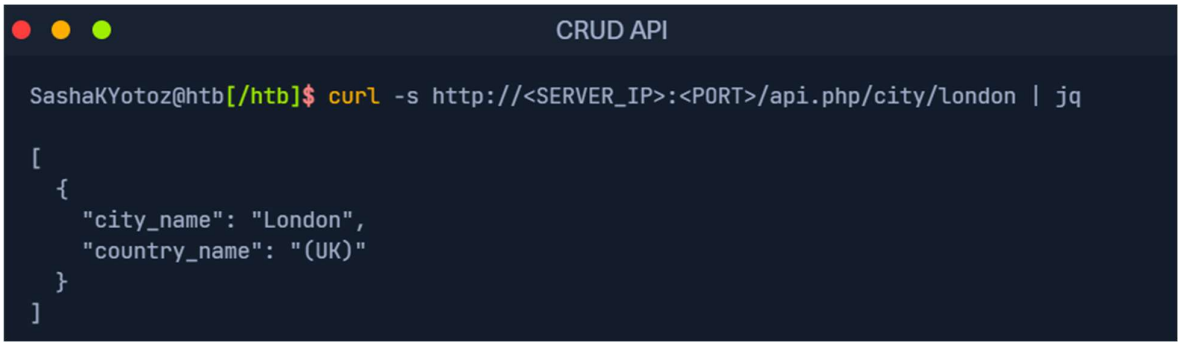
The first thing we will do when interacting with an API is reading data. As mentioned earlier, we can simply specify the table name after the API (e.g. /city) and then specify our search term (e.g. /london), as follows:

CRUD API

```
SashaKYotoz@htb[/htb]$ curl http://<SERVER_IP>:<PORT>/api.php/city/london

[{"city_name":"London","country_name":"(UK)"}]
```

We see that the result is sent as a JSON string. To have it properly formatted in JSON format, we can pipe the output to the jq utility, which will format it properly. We will also silence any unneeded cURL output with -s, as follows:



Create

To add a new entry, we can use an HTTP POST request, which is quite similar to what we have performed in the previous section. We can simply POST our JSON data, and it will be added to the table. As this API is using JSON data, we will also set the Content-Type header to JSON, as follows:

Update

As mentioned at the beginning of the section, PUT is used to update API entries and modify their details, while DELETE is used to remove a specific entity

Note: *The HTTP PATCH method may also be used to update API entries instead of PUT. To be precise, PATCH is used to partially update an entry (only modify some of its data "e.g. only city_name"), while PUT is used to update the entire entry*

Using PUT is quite similar to POST in this case, with the only difference being that we have to specify the name of the entity we want to edit in the URL

Delete

We simply specify the city name for the API and use the HTTP DELETE method, and it would delete the entry, as follows: