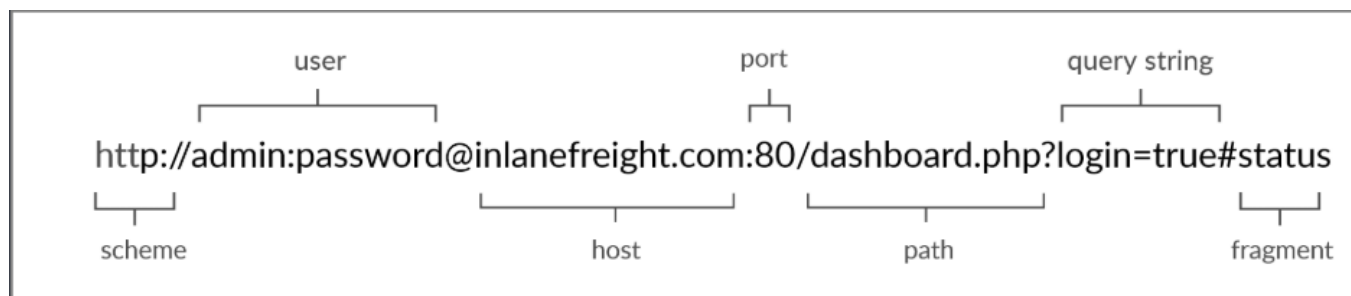


Web Request (intro)

HyperText Transfer Protocol (HTTP)

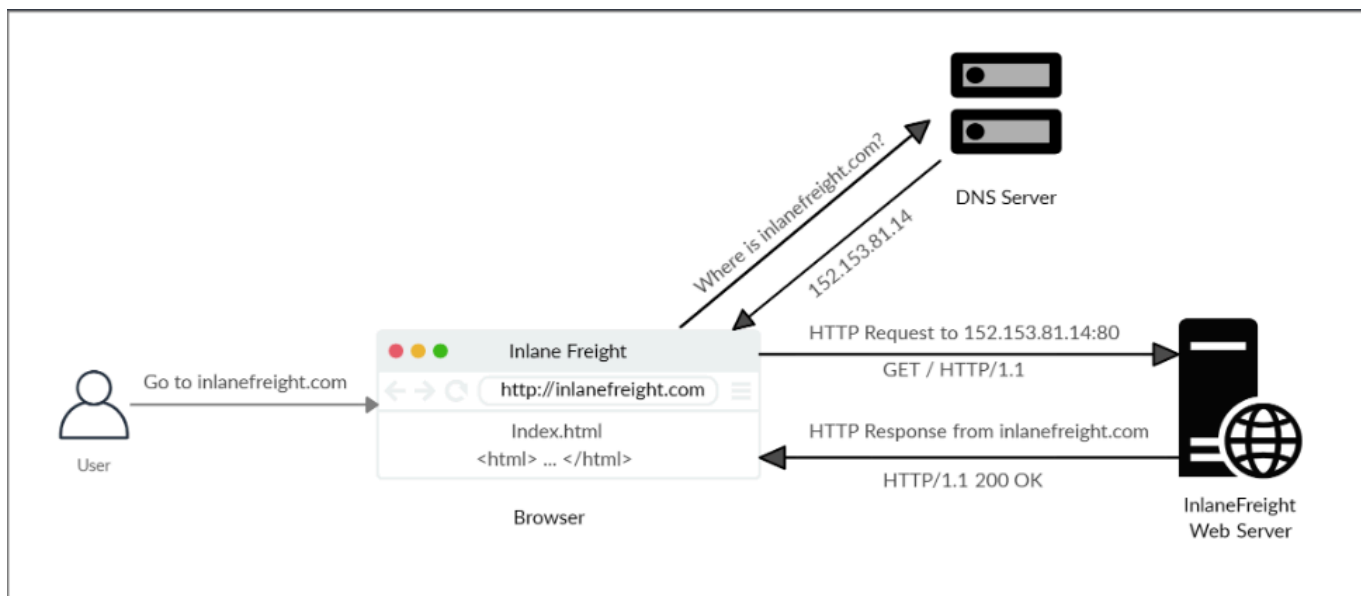
La comunicación HTTP consta de un **cliente y un servidor**. El cliente solicita un recurso al servidor. El servidor procesa las solicitudes y devuelve el recurso solicitado. El puerto predeterminado para la comunicación HTTP es el **80**, aunque puede cambiarse a cualquier otro puerto según la configuración del servidor web.

URL



Component	Example	Description
Scheme	<code>http://https://</code>	This is used to identify the protocol being accessed by the client, and ends with a colon and a double slash (<code>://</code>)
User Info	<code>admin:password@</code>	This is an optional component that contains the credentials (separated by a colon <code>:</code>) used to authenticate to the host, and is separated from the host with an at sign (<code>@</code>)
Host	<code>inlanefreight.com</code>	The host signifies the resource location. This can be a hostname or an IP address
Port	<code>:80</code>	The Port is separated from the Host by a colon (<code>:</code>). If no port is specified, http schemes default to port 80 and https default to port 443
Path	<code>/dashboard.php</code>	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. <code>index.html</code>).
Query String	<code>?login=true</code>	The query string starts with a question mark (<code>?</code>), and consists of a parameter (e.g. <code>login</code>) and a value (e.g. <code>true</code>). Multiple parameters can be separated by an ampersand (<code>&</code>).
Fragments	<code>#status</code>	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



Nota: Nuestros navegadores suelen buscar primero los registros en el archivo local **'/etc/hosts'** y, si el dominio solicitado no existe en él, contactan con otros servidores DNS. Podemos usar **'/etc/hosts'** para agregar manualmente registros para la resolución DNS, añadiendo la IP seguida del nombre de dominio.

cURL

Observamos que cURL **no renderiza el código HTML/JavaScript/CSS**, a diferencia de un navegador web, sino que lo imprime sin procesar. Sin embargo, como evaluadores de penetración, nos interesa principalmente el contexto de solicitud y respuesta, que suele ser mucho **más rápido y práctico** que un navegador web.

También podemos usar cURL para descargar una página o un archivo y generar el contenido en un archivo usando el indicador **-O**. Si queremos especificar el nombre del archivo de salida, podemos usar el indicador **-o** e indicar el nombre. De lo contrario, podemos usar **-O** y cURL usará el nombre del archivo remoto, utilizamos **-s** para silenciar el status, como se indica a continuación:

```
curl -O -s inlane freight.com/index.html
```

Hypertext Transfer Protocol Secure (HTTPS)

Una de las principales desventajas de HTTP es que todos los datos se transfieren en texto plano. Esto significa que cualquier persona entre el origen y el destino puede realizar un ataque de intermediario (**MiTM**) para acceder a los datos transferidos.

Para solucionar esto se creó protocolo HTTPS, donde el tráfico va cifrado.

HTTPS Overview

En HTTP:

74.573774918	192.168.0.108	192.168.0.108	TCP	7640386 → 80 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1
84.573794134	192.168.0.108	192.168.0.108	TCP	7680 → 40386 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1
94.573806187	192.168.0.108	192.168.0.108	TCP	6840386 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=280780439
104.573966701	192.168.0.108	192.168.0.108	HTTP	640POST /login.php HTTP/1.1 (application/x-www-form-urlencoded)
114.573985767	192.168.0.108	192.168.0.108	TCP	6880 → 40386 [ACK] Seq=1 Ack=573 Win=65024 Len=0 TSval=280780439

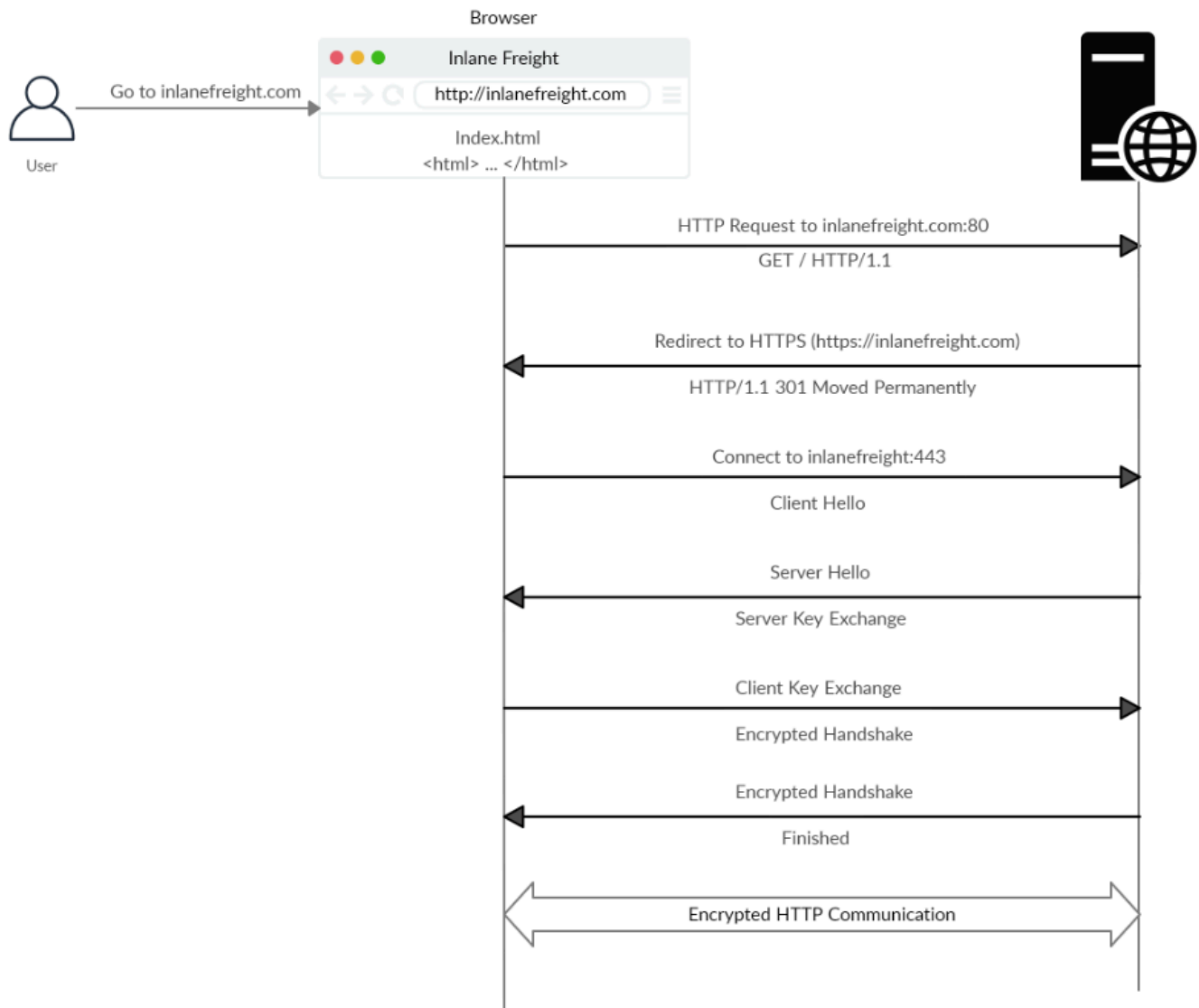
Frame 10: 640 bytes on wire (5120 bits), 640 bytes captured (5120 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 192.168.0.108, Dst: 192.168.0.108
Transmission Control Protocol, Src Port: 40386, Dst Port: 80, Seq: 1, Ack: 1, Len: 572
Hypertext Transfer Protocol
HTML Form URL Encoded: application/x-www-form-urlencoded
Form item: "username" = "admin"
Key: username
Value: admin
Form item: "password" = "password"
Key: password
Value: password

En HTTPS se utiliza TLS para cifrar:

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 TSval=280780439
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 TSval=280780439
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 TSval=280780439

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...

HTTPS Flow



cURL for HTTPS

cURL debería gestionar automáticamente todos los estándares de comunicación HTTPS, realizar un protocolo de enlace seguro y, posteriormente, cifrar y descifrar los datos automáticamente. Sin embargo, si contactamos con un sitio web con un certificado SSL no válido o desactualizado, cURL, por defecto, no procederá con la comunicación para protegerse contra los ataques MITM mencionados anteriormente.

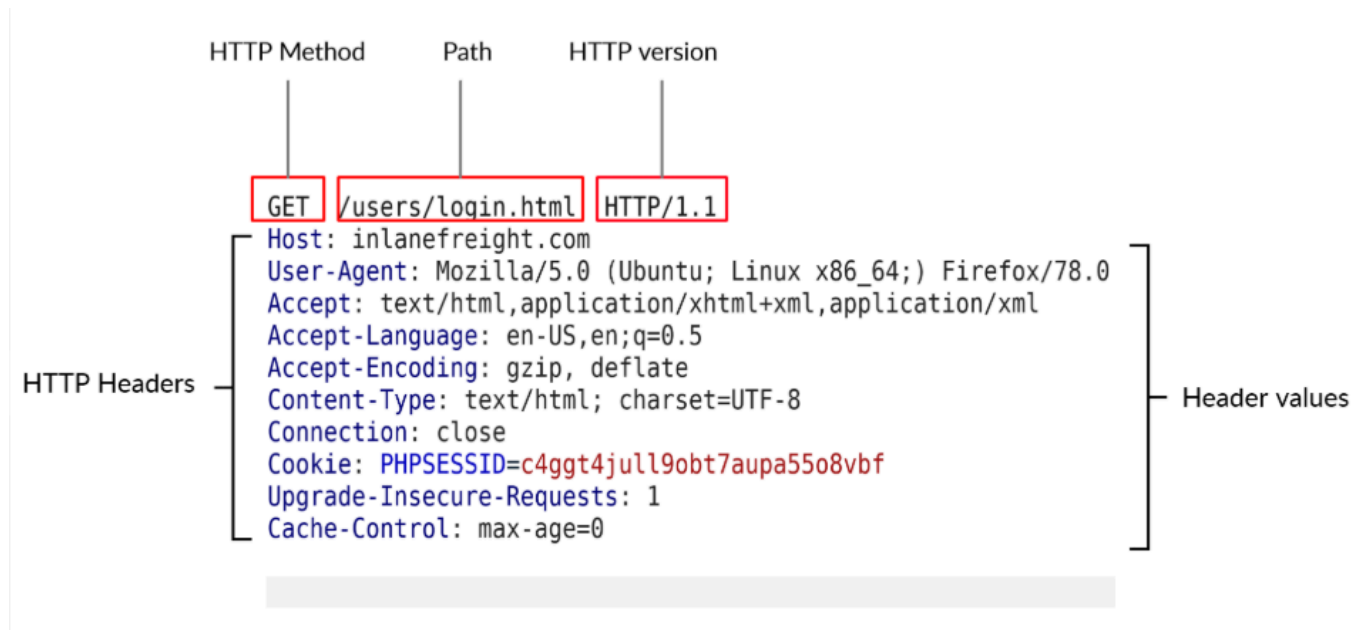
```
curl https://inlanefreight.com
```

```
curl: (60) SSL certificate problem: Invalid certificate chain
More details here: https://curl.haxx.se/docs/sslcerts.html
...SNIP...
```

Podemos omitir el pedir el certificado con el parámetro `-k`.

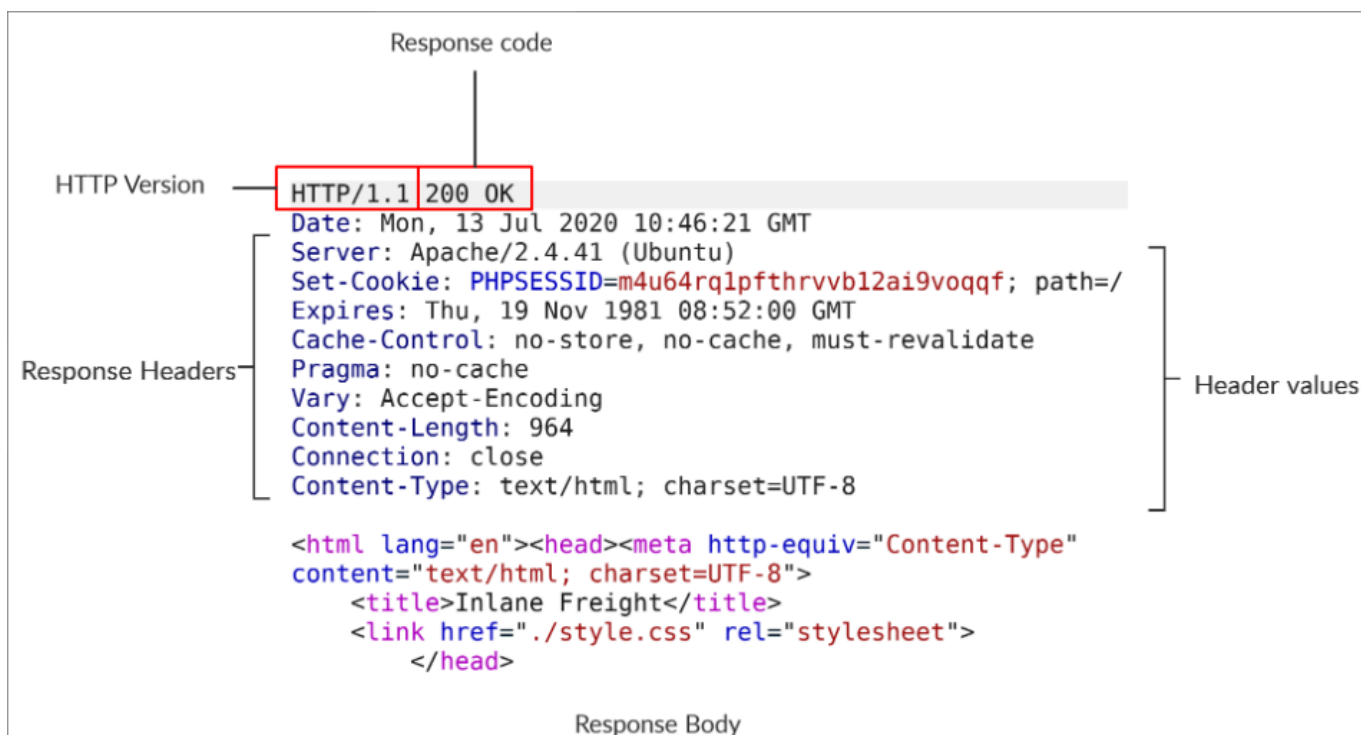
HTTP Requests and Responses

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 Response



HTTP Headers

Estos encabezados HTTP (vistos anteriormente) transmiten información entre el cliente y el servidor. Algunos encabezados solo se usan con solicitudes o respuestas, mientras que otros son comunes a ambas.

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

General Headers

Describe el mensaje en lugar de su contenido.

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

Describe el contenido.

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

Se utilizan en una solicitud HTTP y no se relacionan con el contenido del mensaje.

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

Se utilizan en una respuesta HTTP y no se relacionan con el contenido.

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

Clase de encabezados de respuesta que se utilizan para especificar ciertas reglas y políticas.

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.

HTTP Methods and Codes

Request Methods

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

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.

Algunos ejemplo típicos:

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

Al visitar una URL, nuestros navegadores realizan una solicitud GET por defecto para obtener los recursos remotos alojados en esa URL. Una vez que el navegador recibe la página inicial que solicita, puede enviar otras solicitudes mediante diversos métodos HTTP. Esto se puede observar en la pestaña "**Red**" de las herramientas de desarrollo del navegador, como se vio en la sección anterior.

HTTP Basic Auth

Podemos **loguearnos** desde la terminal a un servicio con **curl** tal que:

```
curl -u admin:admin http://<SERVER_IP>:<PORT>/
```

HTTP Authorization Header

Podemos acceder sin credenciales si nos autentificamos con el token de autorización (**Authorization: Basic YWRtaW46YWRtaW4=**), lo especificamos con el parámetro **-H**.

```
curl -H 'Authorization: Basic YWRtaW46YWRtaW4=' http://<SERVER_IP>:
<PORT>/
```

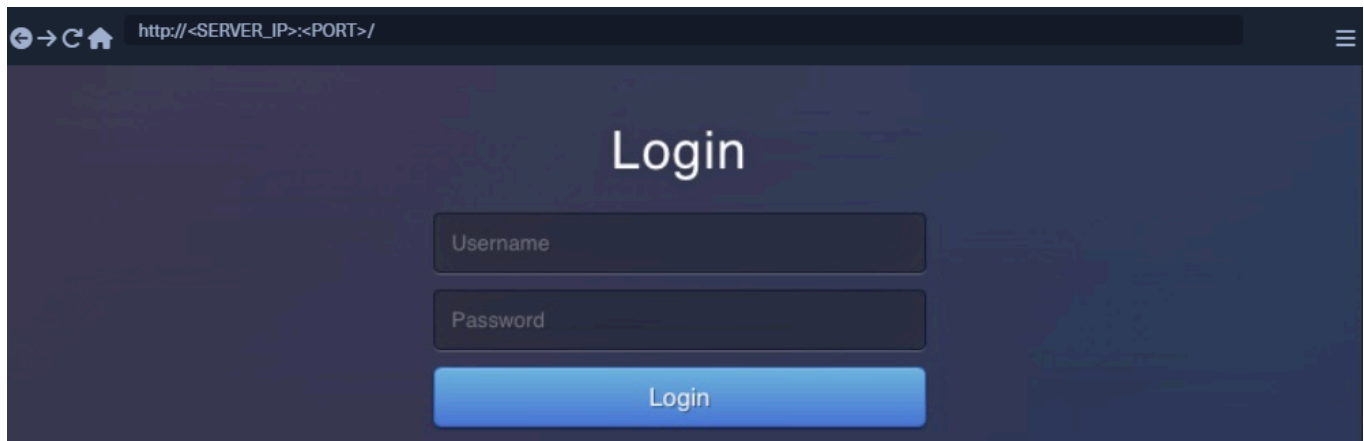
Para utilizar una petición concreta usaremos:

```
curl -u <username>:<password> 'http://example.com/file.php?serch=flag'
```

POST

Vimos cómo las aplicaciones web pueden usar las solicitudes GET para funciones como la búsqueda y el acceso a páginas. Sin embargo, cuando las aplicaciones web necesitan transferir archivos o mover los parámetros del usuario desde la URL, utilizan solicitudes POST.

Login Forms



```
curl -X POST -d 'username=admin&password=admin' http://<SERVER_IP>:
<PORT>/
```

Authenticated Cookies

Esta vez debemos fijarnos en el valor de la cookie de sesión (**Set-Cookie: PHPSESSID=c1nsa6op7vtk7kdis7bcnbadf1**), esto le vemos con el comando:

```
```shell-session
curl -X POST -d 'username=admin&password=admin' http://<SERVER_IP>:
```

```
<PORT>/ -i
```

## JSON Data

---

```
curl -X POST -d '{"search":"london"}' -b
'PHPSESSID=c1nsa6op7vtk7kdis7bcnbadf1' -H 'Content-Type:
application/json' http://<SERVER_IP>:<PORT>/search.php
```

Podemos interactuar con la función de búsqueda directamente sin necesidad de iniciar sesión ni interactuar con el frontend de la aplicación web. Esto puede ser una habilidad esencial al realizar evaluaciones de aplicaciones web o programas de recompensas por errores, ya que es mucho más rápido probar aplicaciones web de esta manera.

## CRUD API

---

En las secciones anteriores, vimos ejemplos de una aplicación web de búsqueda de ciudades que utiliza parámetros PHP para buscar el nombre de una ciudad. En esta sección, analizaremos cómo una aplicación web de este tipo puede utilizar las API para realizar la misma función e interactuaremos directamente con el punto final de la API.

## APIs

---

Existen varios tipos de API. Muchas se utilizan para interactuar con una **base de datos**, de modo que podamos especificar la tabla y la fila solicitadas en nuestra consulta de API y luego usar un método HTTP para realizar la operación necesaria. Por ejemplo, para el endpoint `api.php` de nuestro ejemplo, si quisiéramos actualizar la tabla de ciudades en la base de datos y la fila que actualizaremos tiene como nombre la ciudad de Londres, la URL sería similar a esta:

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

## CRUD

---

Como podemos ver, podemos especificar fácilmente la tabla y la fila en la que queremos realizar una operación mediante estas API. Posteriormente, podemos utilizar diferentes métodos HTTP para realizar distintas operaciones en esa fila. En general, las API realizan cuatro operaciones principales en la entidad de base de datos solicitada:

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

---

Usamos **jq** para que nos lo muestre en el formato correcto:

```
curl http://<SERVER_IP>:<PORT>/api.php/city/london | jq
```

## Create

---

```
curl -X POST http://<SERVER_IP>:<PORT>/api.php/city/ -d
'{"city_name":"HTB_City", "country_name":"HTB"}' -H 'Content-Type:
application/json'
```

Luego con el comando anterior podemos observar como se ha añadido esa nueva entrada:

```
curl -s http://<SERVER_IP>:<PORT>/api.php/city/HTB_City | jq
```

```
[
 {
 "city_name": "HTB_City",
 "country_name": "HTB"
 }
]
```

## Update

---

En este caso, usar PUT **es bastante similar a POST**, con la única diferencia de que **debemos especificar el nombre de la entidad que queremos editar en la URL**; de lo contrario, la API no sabrá qué entidad editar. Por lo tanto, solo tenemos que especificar el nombre de la ciudad en la URL, cambiar el método de solicitud a PUT y proporcionar los datos JSON como hicimos con POST, de la siguiente manera:

```
curl -X PUT http://<SERVER_IP>:<PORT>/api.php/city/london -d
'{"city_name":"New_HTB_City", "country_name":"HTB"}' -H 'Content-Type:
application/json'
```

## DELETE

---

```
curl -X DELETE http://<SERVER_IP>:<PORT>/api.php/city/New_HTB_City
```