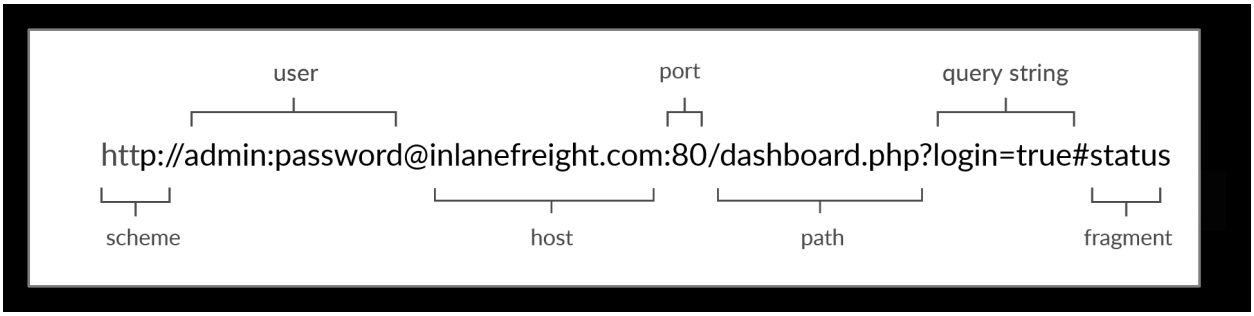
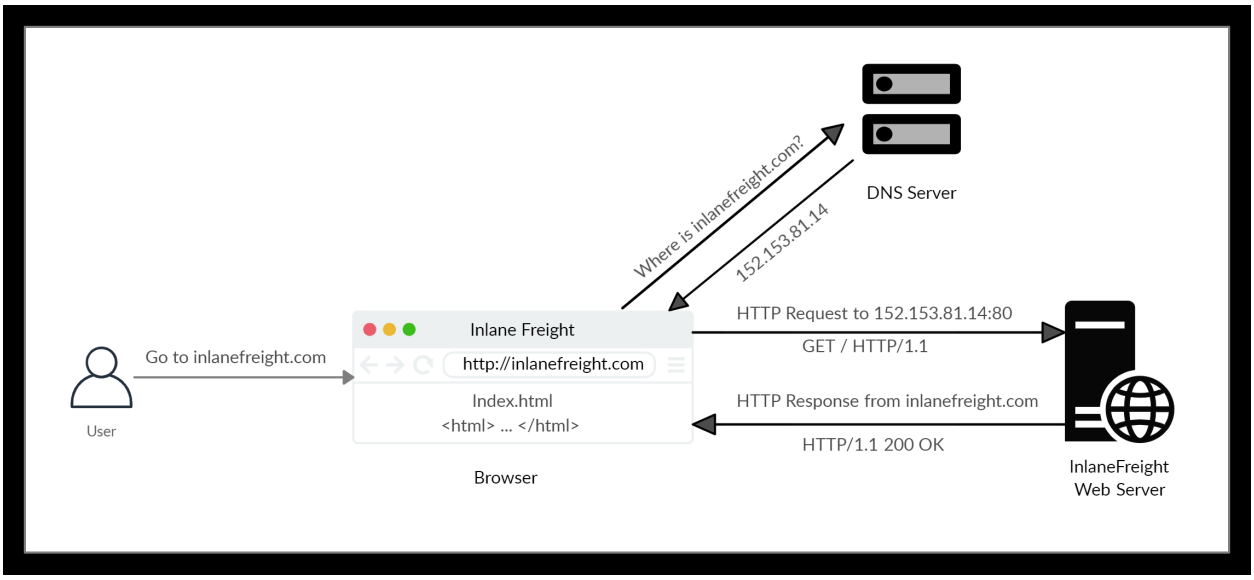
Web Requests

HyperText Transfer Protocol (HTTP)

Most internet communications are made with web requests through the HTTP protocol. 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`.



HTTP Flow



cURL

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

```
OmVg@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.
```

Questions

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

Target: Click here to spawn the target system!

To get the flag, use cURL to download the file returned by '/download.php' in the above server.

Ans: HTB{64$!c_cURL_u$3r}

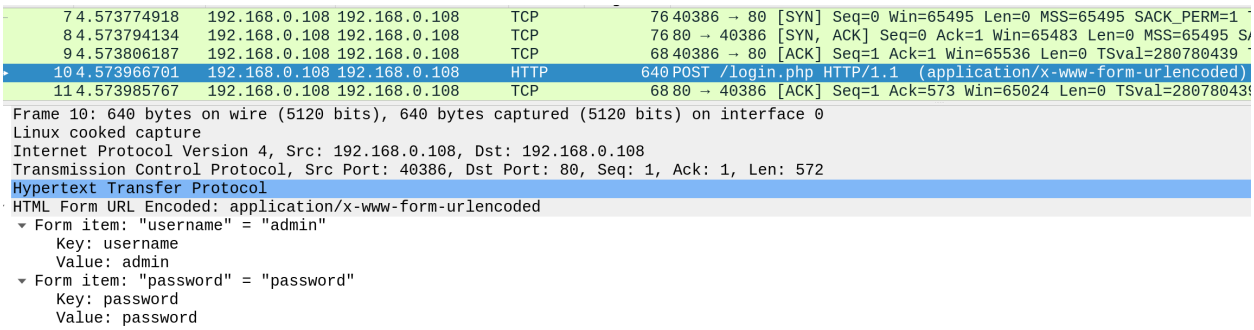
Hypertext Transfer Protocol Secure (HTTPS)

One of the significant drawbacks of HTTP is that all data is transferred in clear text. This means that anyone between the source and the destination can perform a Man-in-the-middle (MiTM) attack to view the transferred data.

To counter this issue, the HTTPS (HTTP Secure) protocol was created, in which all communications are transferred in an encrypted format, so even if a third party does intercept the request, they would not be able to extract the data from it. For this reason, HTTPS has become the mainstream scheme for websites on the internet, and HTTP is being phased out, and soon most web browsers will not allow visiting HTTP websites.

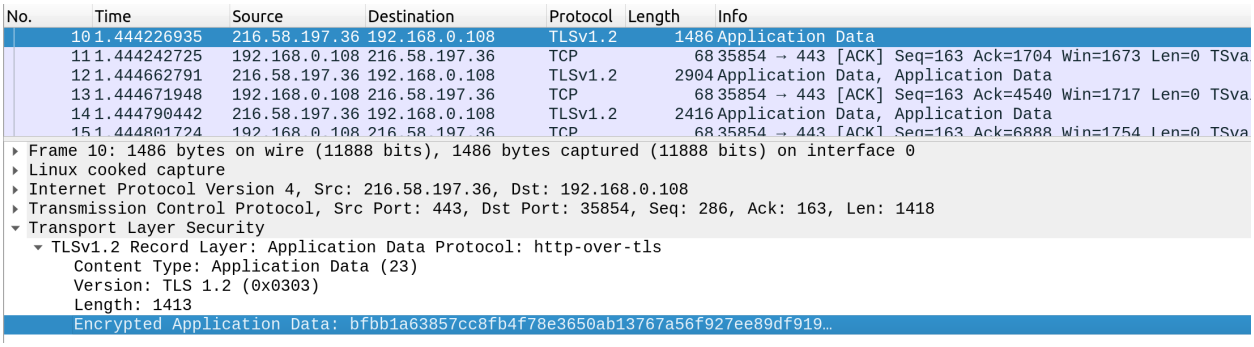
HTTPS Overview

If we examine an HTTP request, we can see the effect of not enforcing secure communications between a web browser and a web application. For example, the following is the content of an HTTP login request:



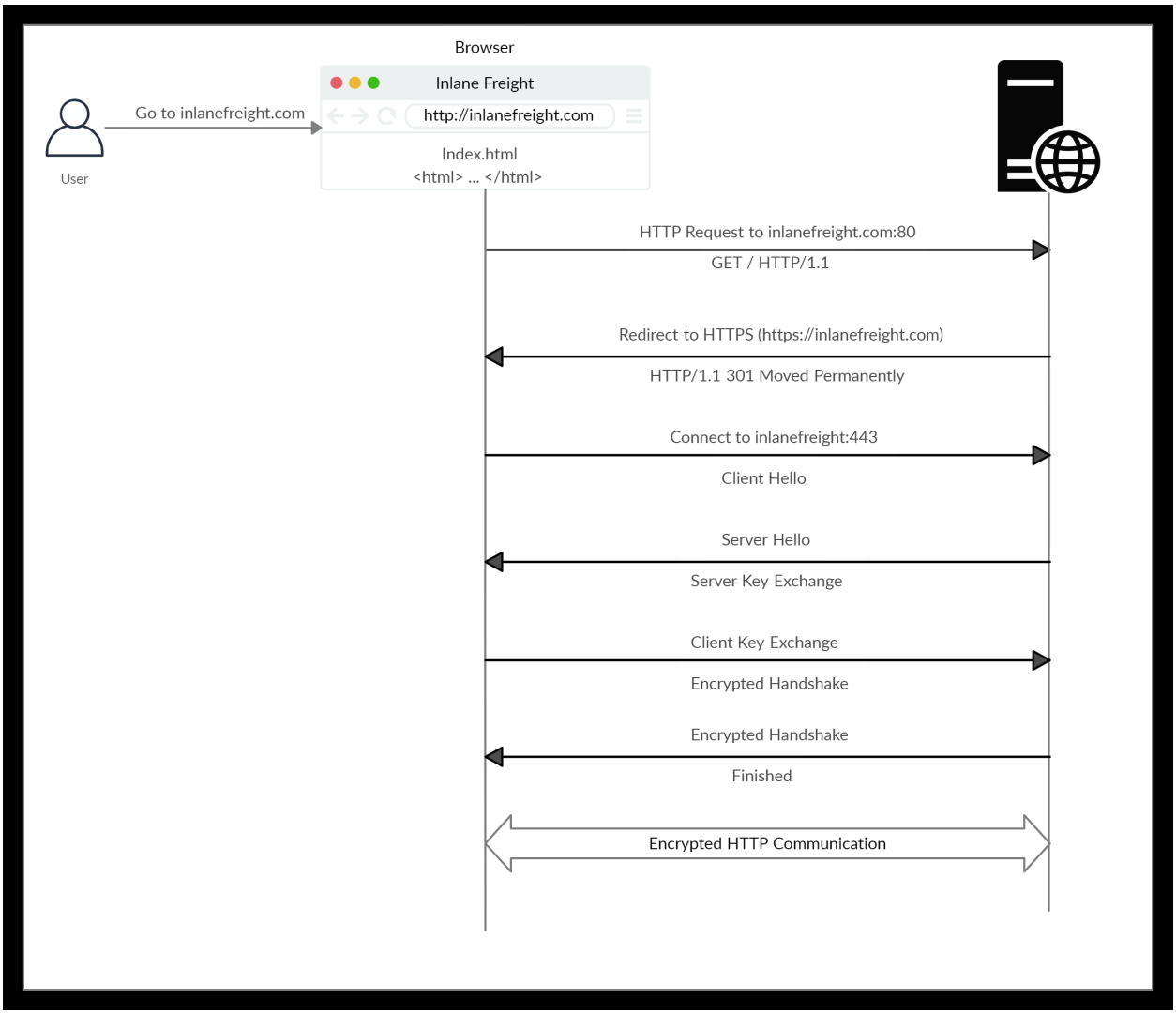
We can see that the login credentials can be viewed in clear-text. This would make it easy for someone on the same network (such as a public wireless network) to capture the request and reuse the credentials for malicious purposes.

In contrast, when someone intercepts and analyzes traffic from an HTTPS request, they would see something like the following:



As we can see, the data is transferred as a single encrypted stream, which makes it very difficult for anyone to capture information such as credentials or any other sensitive data.

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, which is the unencrypted HTTP protocol. The server detects this and redirects the client to secure HTTPS port `443` instead. This is done via the `301 Moved Permanently` response code, which we will discuss in an upcoming section.

Next, the client (web browser) 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. The client verifies the key/certificate and sends one of its

own. After this, an encrypted handshake is initiated to confirm whether the encryption and transfer are working correctly.

Once the handshake completes successfully, normal HTTP communication is continued, which is encrypted after that. This is a very high-level overview of the key exchange, which is beyond this module's scope.

cURL for HTTPS

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

To skip the certificate check with cURL, we can use the `-k` flag

HTTP Requests and Responses

HTTP communications mainly consist of an HTTP request and an HTTP response. An HTTP request is made by the client (e.g. cURL/browser), and is processed by the server (e.g. webserver). The requests contain all of the details we require from the server, including the resource (e.g. URL, path, parameters), any request data, headers or options we specify, and many other options we will discuss throughout this module.

Once the server receives the HTTP request, it processes the requests and responds by sending the HTTP response, which contains the response code, as discussed in a later section, and may contain the resource data if the requester had access to it.

Questions

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

Target: 178.128.163.152:32131

Time Left: 79 minutes

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

Ans: GET

Send a GET request to the above server, and read the response headers to find the version of Apache running on the server? (answer format: X.Y.ZZ)

Ans: 2.4.41

HTTP Headers

We have seen examples of HTTP requests and response headers in the previous section. Such HTTP headers pass information between the client and the server. Some headers are only used with either requests or responses, while some other general headers are common to both.

Headers can have one or multiple values, appended after the header name and separated by a colon. 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

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

Similar to general headers, 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.
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 denotes the encoding standard, such as UTF-8.
Boundary	`boundary="b4e4fbd93540"`	Acts as a maker 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

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

Header	Example	Description
`Host`	`Host: www.inlanefreight.com`	Used to specify the host being queried for the resource. This can be a domain name or an IP address. HTTP servers can be configured to host different websites, which are revealed based on the hostname. This makes the host header an important enumeration target, as it can indicate the existence of other hosts on the target server.
`User-Agent`	`User-Agent: curl/7.77.0`	The `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 contains multiple media types separated by commas. The `*/*` value signifies that all media types are accepted.
`Cookie`	`Cookie: PHPSESSID=b4e4fbd93540`	Contain 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

Finally, we have Security Headers.

With the increase in the variety of browsers and web-based attacks, defining certain headers that enhanced security was necessary. 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.