Basics of HTTP/HTTPS

Understanding the Basic Differences:

HTTP and HTTPS are both protocols used for transferring data across the web, but they have fundamental differences, particularly around **security**:

- **HTTP**:
  - **Data Transmission**: Plaintext.
  - **Security**: No encryption, making the data vulnerable to interception (man-in-the-middle attacks).
  - **Port**: Typically operates on port 80.
  - **Use Case**: Suitable for non-sensitive data where encryption isn't necessary, but outdated in modern web practices due to security concerns.
- **HTTPS**:
  - **Data Transmission**: Encrypted using SSL/TLS.
  - **Security**: Provides confidentiality (data is encrypted), integrity (data can't be altered without detection), and authentication (validates the server identity).
  - **Port**: Typically operates on port 443.
  - **Use Case**: Preferred for all websites, especially those handling sensitive information like banking or personal data. It's a modern standard.

2. Main Differences (Focusing on Security):

- **Encryption**:
  - HTTP does not encrypt the data, meaning anyone who intercepts the data can read it.
  - HTTPS encrypts the data using SSL/TLS, making it unreadable to unauthorized users.
- **Data Integrity**:
  - In HTTP, data can be altered during transmission without being noticed.
  - HTTPS ensures data integrity, meaning that if data is tampered with, it will be detected.

- **Authentication**:
    - HTTP does not validate the identity of the server.
    - HTTPS requires an SSL/TLS certificate issued by a trusted Certificate Authority (CA), ensuring that the client is communicating with the intended server and not an imposter.
- **SEO and Browser Trust**:
    - Sites using HTTP are flagged as "not secure" by modern browsers.
    - HTTPS increases trust (with a padlock symbol in the address bar) and can boost SEO rankings.

3. Optional: Using a Packet Sniffer (Wireshark):

If you decide to explore this further by observing traffic in Wireshark, here's what you might see:

- **HTTP Traffic**:
    - All the data will be in plaintext, and you'll be able to view the entire content of the request and response, including sensitive information like login details, passwords, etc.
- **HTTPS Traffic**:
    - The data will appear as encrypted, and you won't be able to read the content directly because it's protected by SSL/TLS. However, you can still see some metadata like the server IP and the encrypted packet size.

Understanding the structure of HTTP requests and responses is crucial for web development and networking. Here's a breakdown of how you can approach it:

1. Open Developer Tools

- **Visit a website**: Open any website in your browser.
- **Inspect the page**: Right-click on the page and choose “Inspect” or “Inspect Element.”
- **Navigate to the Network Tab**: Open the browser’s Developer Tools (usually by pressing `F12` or right-clicking to “Inspect”), then navigate to the **Network** tab.

2. Reload the Page

- **Reload**: Once the Network tab is open, reload the page by pressing `F5` or clicking the refresh button. This shows all the network requests being made by the browser.

3. First Request: HTTP Request Structure

- **First Request**: The first request listed is typically for the main HTML document (or the URL you visited).
- **Click on it**: When you click on this request, you’ll see detailed information about it in the right-hand panel. This includes the headers, method, and status code.

4. Explore the Headers (HTTP Request)

Key Sections of an HTTP Request:

- **Request Method**: The method defines what action the server should take (e.g., `GET`, `POST`, `PUT`, `DELETE`).
    - `GET`: Requests data from the server (e.g., HTML, images).
    - `POST`: Sends data to the server (e.g., submitting a form).
    - `PUT`, `DELETE`: Used for modifying or deleting resources.
- **Request URL/Path**: Shows the exact URL or resource the request is being made to. For example, `/index.html`.
- **HTTP Version**: The version of HTTP protocol (e.g., `HTTP/1.1` or `HTTP/2`).
- **Headers**:
    - **Host**: The domain being requested (e.g., `example.com`).
    - **User-Agent**: Information about the client making the request (browser, operating system).

    - **Accept**: The media types the client can understand (e.g., `text/html`, `application/json`).
    - **Cookies**: Any cookies stored by the browser for the domain.
    - **Referer**: The page that referred the request, if any.
- **Payload/Body** (optional): Only for `POST` or `PUT` requests where data is being sent (e.g., form data).

5. Explore the Headers (HTTP Response)

Key Sections of an HTTP Response:

- **Status Code**: The HTTP status code tells you how the request was handled:
    - `200 OK`: The request was successful.
    - `404 Not Found`: The requested resource was not found.
    - `500 Internal Server Error`: The server encountered an error.
- **Response Headers**:
    - **Content-Type**: Describes the type of content being returned (e.g., `text/html`, `application/json`).
    - **Content-Length**: The size of the response body in bytes.
    - **Cache-Control**: Specifies how the response should be cached by the browser.
    - **Set-Cookie**: Sets cookies to be stored on the client.
- **Response Body**: Contains the actual content returned by the server, such as HTML, JSON, or image data. You may view this in the **Preview** or **Response** tab in the developer tools.

6. Additional Observations

- **Other Requests**: Apart from the main HTML request, you'll see requests for other resources like CSS, JavaScript, images, and fonts.
- **Filtering Requests**: You can filter these requests by type (e.g., `XHR` for API calls, `CSS`, `JS`) to focus on specific types of traffic.

Exploring HTTP Methods and Status Codes:

Common HTTP Methods:

1. **GET**
   - **Usage**: Used to request data from a server. This method retrieves information, such as a web page, API data, or an image, without changing the state of the resource on the server.
   - **Example**: A user requests to view the home page of a website, and the server sends the content of the page.
2. **POST**
   - **Usage**: Used to submit data to the server, such as form data or an API request, to create or update a resource. This method changes the state of the server.
   - **Example**: A user submits a login form on a website, and the server processes the credentials to authenticate the user.
3. **PUT**
   - **Usage**: Used to update or replace an existing resource on the server. If the resource doesn’t exist, it may create a new one.
   - **Example**: Updating a user's profile information, such as changing the user's email address in a database.
4. **DELETE**
   - **Usage**: Used to delete a resource from the server. It removes the specified resource from the server permanently.
   - **Example**: A user deletes their account from a website, and the server removes all the associated data.

Common HTTP Status Codes:

1. **200 OK**
   - **Description**: The request was successful, and the server returned the requested resource.
   - **Scenario**: A user successfully loads a web page, and the browser receives the requested HTML and resources.
2. **201 Created**
   - **Description**: The request has been fulfilled, and a new resource has been created.
   - **Scenario**: After submitting a form to create a new account, the server
   - confirms that the account was successfully created.

3. **400 Bad Request**
   - **Description**: The server could not understand the request due to invalid syntax or malformed input.
   - **Scenario**: A user tries to submit a form with missing or invalid data (e.g., entering an incorrectly formatted email address).
4. **401 Unauthorized**
   - **Description**: The client must authenticate itself to get the requested response. This often occurs when credentials are missing or incorrect.
   - **Scenario**: A user tries to access a restricted area of a website without logging in.
5. **404 Not Found**
   - **Description**: The server could not find the requested resource.
   - **Scenario**: A user clicks a broken link, and the server responds that the requested page does not exist.