WEB BASICS

1.1 What is a Website?

A **website** is a collection of **web pages** (HTML, CSS, JS files, images, etc.) stored on a **web server** that can be accessed through the **internet** using a **browser**.

👉 Example: When you type `www.google.com` in your browser, your computer requests Google’s server, gets back the response (HTML, CSS, JS), and displays it as a webpage.

1.2 Structure of a Website

A website works through **3 main layers**:

1. **Client-Side (Frontend)** – What the user sees.

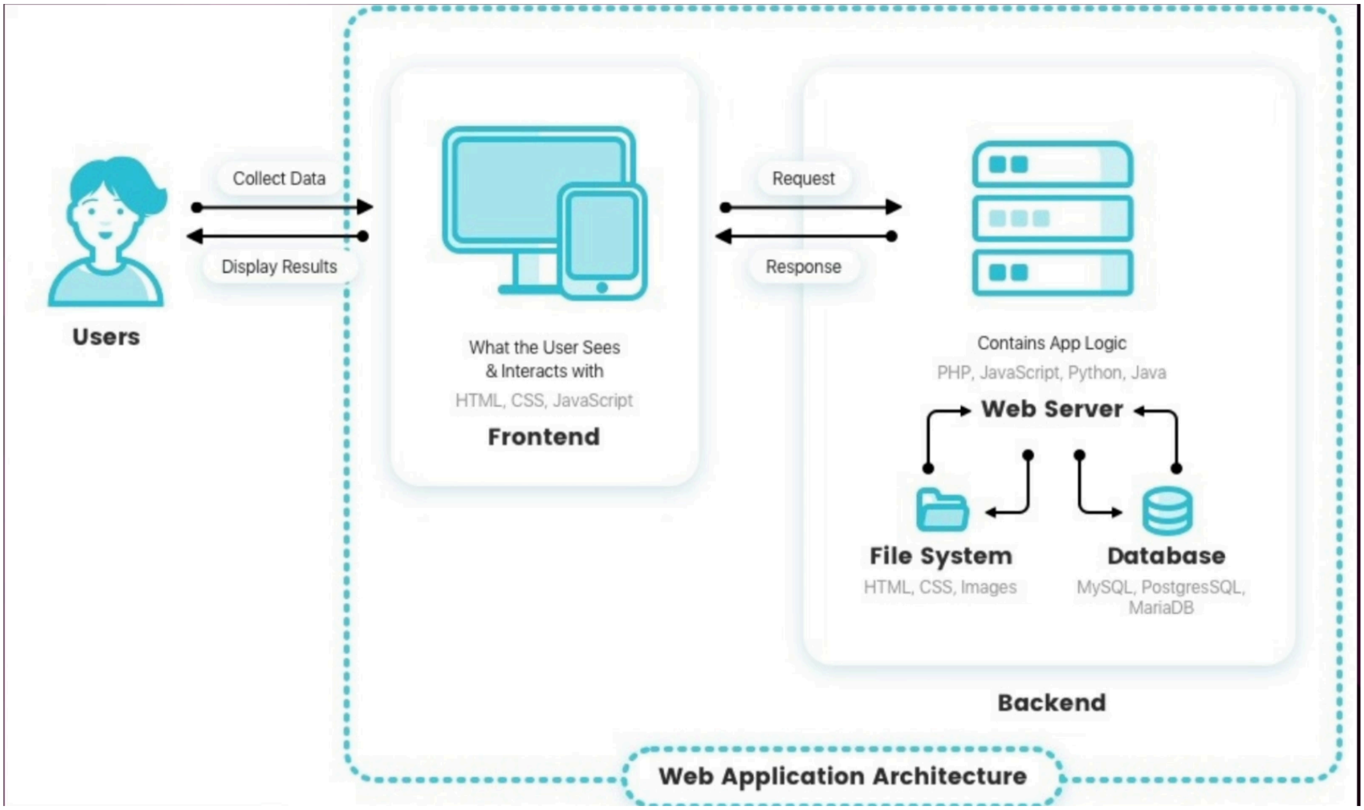
    - HTML (structure: text, images, links)
    - CSS (style: colors, fonts, layout)
    - JavaScript (interactivity: buttons, forms, dynamic content)

    ⚡ Runs in your browser (Chrome, Firefox, etc.).
2. **Server-Side (Backend)** – Logic + data processing.
    - Written in languages like PHP, Python (Django/Flask), Node.js, Java, Ruby.
    - Handles login, payments, search queries, etc.
    - Connects with the **database**.
3. **Database** – Stores data.
    - MySQL, PostgreSQL, MongoDB, etc.
    - Stores user accounts, passwords, posts, transactions.

1.3 How a Website Works (Step by Step)

1. **We enter a website URL** (e.g., `facebook.com`) in your browser.
2. **DNS Lookup happens** – Your computer asks a DNS server:
   “What is the IP address of `facebook.com`?”

3. **Browser connects to server** – Using that IP, your browser connects to Facebook's web server over the internet.
4. **HTTP/HTTPS request is sent** – Your browser sends a request (like: "Give me the homepage").
5. **Server processes request** – The web server checks databases, runs backend code, and prepares the webpage.
6. **Response is sent back** – The server sends HTML, CSS, JS files (and sometimes JSON/XML if API calls).
7. **Browser renders page** – Your browser takes those files, builds the webpage, executes JavaScript, and displays it to you.



1.4 Important Technical Details

Here are the **building blocks of how websites work**:

1. Frontend (Client Side)

- Languages: **HTML (structure), CSS (style), JS (functionality)**.
- Runs on your browser.
- Responsible for how the website looks and basic interactions.

2. Backend (Server Side)

- Languages: **Python (Django/Flask), PHP, Java (Spring), Node.js, Ruby, .NET**, etc.
- Handles logic, authentication, databases, APIs.
- Example: When you log in, backend checks if your password matches.

3. Databases

- Store data: MySQL, PostgreSQL, MongoDB, Oracle, etc.
- Example: Your username and password are stored (hopefully hashed).

4. DNS (Domain Name System)

- Converts domain names (`facebook.com`) into IP addresses (`157.240.1.35`).
- Like a phonebook of the internet.

5. HTTP / HTTPS

- **HTTP** = Hypertext Transfer Protocol (insecure).
- **HTTPS** = HTTP + TLS/SSL encryption (secure).
- Requests have **methods**:
  - `GET` → Fetch data
  - `POST` → Send data
  - `PUT` / `PATCH` → Update data
  - `DELETE` → Remove data

6. Web Servers

- Handle client requests and serve responses.
- Examples: Apache, Nginx, Microsoft IIS.

7. APIs (Application Programming Interfaces)

- Allow apps/websites to talk to each other.
- Example: A weather website fetching live weather data from an API.

8. Cookies & Sessions

- **Cookie** = Small piece of data stored in the browser.
- **Session** = Server-side record linked with session ID.
- Used for logins, preferences, tracking.
  👉 This is why *session hijacking* is a real attack.

9. Client vs. Server Execution

- Client = Browser executes **JavaScript**.
- Server = Executes backend code, interacts with DB.
- Example:
  - Typing a message → frontend sends request → backend saves in DB → other client fetches via API.

---

2. What is DNS?

DNS = **Domain Name System** → It's like the **phonebook of the internet**.

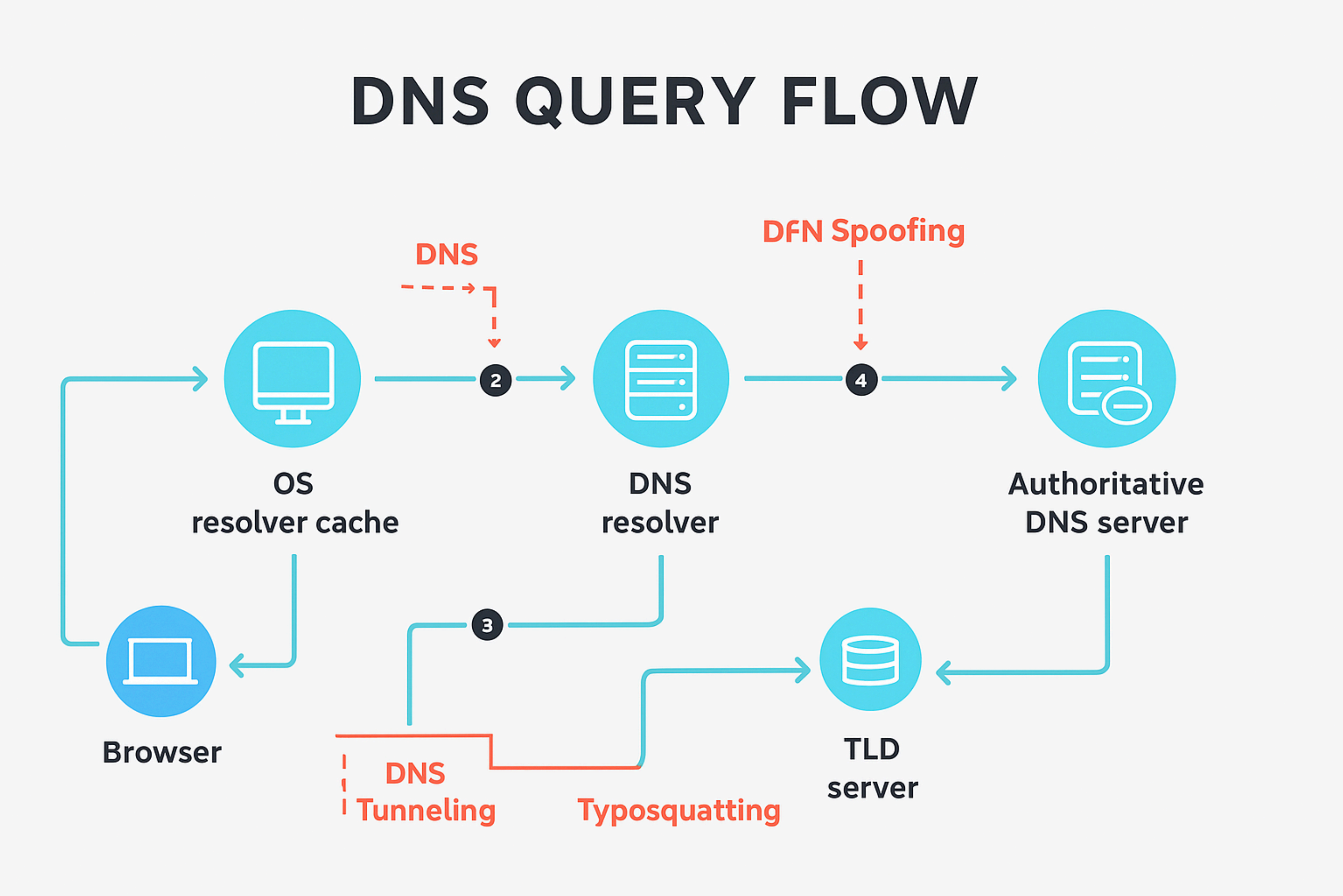
- You type `www.facebook.com` → DNS translates it to an IP address (`157.240.1.35`).
- Without DNS, you'd have to remember IPs for every website.

2.1 How DNS Works (Step by Step)

Example: You type `www.example.com` in the browser.

1. **Browser Cache**
   - First, browser checks if it already knows the IP (from recent visits).
2. **OS Resolver Cache**
   - If browser doesn't have it, it asks the OS (computer's local DNS cache).
3. **DNS Resolver (ISP / Custom)**
   - If OS doesn't know, it asks the **recursive resolver** (usually provided by your ISP, or public like Google `8.8.8.8` , Cloudflare `1.1.1.1`).
4. **Root Server**
   - If resolver doesn't know, it asks a **root DNS server**.
   - Root says: "I don't know `example.com` , but I know where `.com` TLD servers are."
5. **TLD Server (.com, .org, .net, etc.)**
   - The resolver asks the `.com` TLD server.
   - TLD server replies: "Ask the authoritative server for `example.com` ."
6. **Authoritative DNS Server**
   - The authoritative server stores the actual record for `example.com` .
   - It replies: "The IP of `www.example.com` is `93.184.216.34` ."

7. **Response Cached & Sent Back**
    - Resolver caches the answer for future use.
    - Browser gets IP → makes a direct connection to the website.



Security Threats Shown in this Diagram :-

- **DNS Spoofing (Step 2 & 4)**
  Attacker injects fake records → browser redirected to malicious site.
- **DNS Tunneling (Step 3)**
  DNS used to smuggle data (C&C communication or data theft).
- **Typosquatting (Step 3)**
  Fake domains registered (e.g., `g00gle.com`) to trick users.

✅ In short:

- The **blue arrows** show the normal DNS resolution flow.
- The **red labels** highlight **attack surfaces** where hackers can manipulate or abuse DNS.

2.2 Domain Name Structure Hierarchy

A **domain name** is split into different levels (from right to left).

a.) TLD (Top-Level Domain)

- The **last part** (rightmost) of a domain name.
- Example: `tryhackme.com` → **.com** is the TLD.

Types of TLDs:

1. **gTLD (Generic Top-Level Domain)**
    - Meant to describe purpose.
    - Examples:
        - `.com` → commercial
        - `.org` → organization
        - `.edu` → education
        - `.gov` → government
    - Now includes new gTLDs like `.online`, `.club`, `.website`, `.biz` (over 2000 available).
2. **ccTLD (Country Code Top-Level Domain)**
    - Represent geographic regions.
    - Examples:
        - `.ca` → Canada
        - `.co.uk` → United Kingdom
        - `.in` → India

---

b.) Second-Level Domain (SLD)

- The part just **before the TLD**.
- Example: `tryhackme.com` → **tryhackme** is the SLD.

Rules for SLDs:

- Max length: **63 characters** (plus TLD).
- Allowed: `a-z`, `0-9`, and `-` (hyphen).
- Restrictions:
    - Cannot start or end with `-`.
    - Cannot use consecutive `--`.

---

c.) Subdomain

- Anything **before the SLD** (separated by a dot).
- Example:
  - `admin.tryhackme.com` → **admin** is a subdomain.
  - `jupiter.servers.tryhackme.com` → multiple subdomains.

Rules for Subdomains:

- Max length: **63 characters** each.
- Overall domain length: **253 characters** max.
- Allowed characters: `a-z`, `0-9`, `-` (with same restrictions as SLD).
- No limit on number of subdomains.

🔹 Example Breakdown

For: `jupiter.servers.tryhackme.com`

- **.com** → TLD
- **tryhackme** → SLD
- **servers** → subdomain
- **jupiter** → sub-subdomain

✅ In short:

- **TLD** = category or country (.com, .org, .in)
- **SLD** = your main domain name (tryhackme)
- **Subdomain** = extra divisions (admin., mail., blog.)

2.3 Important DNS Records (Very Important in Cybersecurity)

Record Type	Meaning	Example
A	IPv4 address	`www.example.com → 93.184.216.34`
AAAA	IPv6 address	`www.example.com → 2606:2800:220:1:248:1893:25c8:1946`
CNAME	Alias of another domain	`mail.example.com → ghs.google.com`
MX	Mail server record	`example.com → mail.protection.outlook.com`
NS	Name servers (who is authoritative)	`ns1.example.com`
TXT	Arbitrary text (SPF, DKIM, DMARC for email security)	`v=spf1 include:_spf.google.com -all`
PTR	Reverse lookup (IP → Domain)	`93.184.216.34 → example.com`
SRV	Service location	Used in VoIP, MS services
SOA	Start of authority (zone info)	Domain admin details

2.4 DNS Caching

- To speed up lookups, DNS uses **caching**.
- Cached at multiple levels: Browser → OS → Resolver → Recursive server.
- Controlled by **TTL (Time To Live)** in seconds.
  Example: TTL = 86400 → record valid for 1 day.

2.5 DNS Security Concerns (Important for Cybersecurity)

1. **DNS Spoofing / Cache Poisoning**
   - Attacker injects fake DNS records into cache → victim visits malicious site instead of real one.
2. **DNS Hijacking**
   - Changing DNS settings at router/ISP level → redirects traffic.
3. **DNS Tunneling**
   - Attackers encode data inside DNS queries → bypass firewalls (used for data exfiltration).
4. **Typosquatting / Domain Squatting**

- Attacker registers `faceboook.com` or `g00gle.com` to trick users.

5. **Amplification Attack (DDoS)**
   - Using open DNS resolvers to flood a target with massive traffic.

2.6 Related Topics We Should Know

1. **Public DNS Services**
   - Google: `8.8.8.8`, `8.8.4.4`
   - Cloudflare: `1.1.1.1`, `1.0.0.1`
   - OpenDNS (Cisco): `208.67.222.222`, `208.67.220.220`
2. **Reverse DNS (rDNS)**
   - Converts IP → Domain.
   - Useful for email servers (spam protection).
3. **DNS over HTTPS (DoH) / DNS over TLS (DoT)**
   - Encrypts DNS queries (prevents ISP snooping / MITM).
4. **Zone Transfer (AXFR)**
   - Authoritative servers replicate records to secondary servers.
   - Attackers can abuse misconfigured DNS servers to dump entire DNS records.
5. **EDNS (Extension DNS)**
   - Extends standard DNS to support more features.

✅ In short:
DNS = **translator between human-friendly domains and machine IPs**, with **records, caching, and security layers**. It's fast, distributed, but also a weak spot if misconfigured.

---

3. What is HTTP / HTTPS?

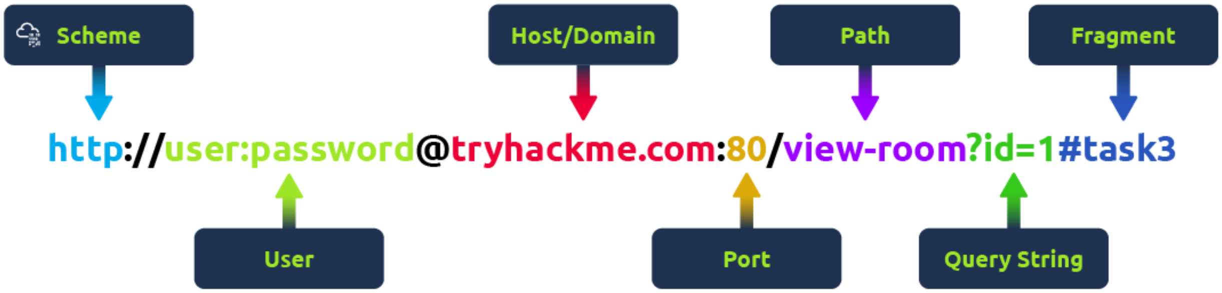
- **HTTP (HyperText Transfer Protocol):**
  - A protocol for transferring web pages between client (browser) and server.
    ❌ Data is sent in plain text (not secure).
  - Developed by Tim Berners-lee and his team between 1989-1991.
- **HTTPS (HTTP Secure):**

- HTTP + **TLS/SSL encryption**.
- ✅ Encrypts communication → secure from eavesdropping & tampering.
- HTTPS data is encrypted so it not only stops people from seeing the data you are receiving and sending, but it also gives you assurances that you're talking to the correct web server and not something impersonating it.

3.1 How HTTP/HTTPS Works (Flow)

1. **When we type a URL**

   - A URL is predominantly an instruction on how to access a resource on the internet.
   - Example:

   - **Scheme:** This instructs on what protocol to use for accessing the resource such as HTTP, HTTPS, FTP (File Transfer Protocol).
   - **User:** Some services require authentication to log in, we can put a username and password into the URL to log in.
   - **Host:** The domain name or IP address of the server you wish to access.
   - **Port:** The Port that you are going to connect to, usually 80 for HTTP and 443 for HTTPS, but this can be hosted on any port between 1 - 65535.
   - **Path:** The file name or location of the resource you are trying to access.
   - **Query String:** Extra bits of information that can be sent to the requested path. For example, /blog?**id=1** would tell the blog path that you wish to receive the blog article with the id of 1.
   - **Fragment:** This is a reference to a location on the actual page requested. This is commonly used for pages with long content and can have a certain part of the page directly linked to it, so it is viewable to the user as soon as they access the page.
2. **DNS Resolution**
   - Browser asks DNS → gets IP address of server.
3. **Connection Establishment**
   - **HTTP** → Browser connects to server's port **80**.

    - **HTTPS** → Browser connects to server’s port **443**.

4. **(HTTPS only) TLS Handshake**
    - Browser and server exchange certificates.
    - Verify server’s identity (via CA – Certificate Authority).
    - Agree on encryption keys for secure session.
5. **HTTP Request Sent**
    - Example:
      `GET /index.html HTTP/1.1 Host: example.com User-Agent: Chrome/120.0 Cookie: sessionid=12345`
    - Request has **method, headers, body (for POST)**.
6. **Server Processes Request**
    - Reads data, queries DB, runs backend code.
7. **HTTP Response Sent Back**
    - Example:
      `HTTP/1.1 200 OK Content-Type: text/html Set-Cookie: sessionid=12345`
    - Includes **status code + headers + body (HTML, JSON, etc.)**.
8. **Browser Renders Page**
    - Uses HTML + CSS + JS to display website.

3.2 HTTP Methods (Very Important in Security)

Method	Purpose	Example
GET	Fetch data	Load a webpage
POST	Send data to server	Login form submission
PUT/PATCH	Update data	Edit profile
DELETE	Remove data	Delete account
HEAD	Fetch headers only	Check server status
OPTIONS	Show allowed methods	Used in CORS

3.3 HTTP Status Codes

Code	Meaning	Example
200	OK (success)	Page loads correctly
301/302	Redirect	Site moved
400	Bad Request	Invalid request
401	Unauthorized	Login needed
403	Forbidden	No access
404	Not Found	Page missing
500	Server Error	Website broken

3.4 Security Difference

Feature	HTTP	HTTPS
Encryption	❌ No	✅ Yes
Port	80	443
Certificate	Not required	SSL/TLS certificate required
Attacks possible	MITM, Sniffing	Harder (data encrypted)

3.5 Common Attacks on HTTP/HTTPS

1. **Man-in-the-Middle (MITM):**
    - In HTTP, attacker can sniff credentials.
2. **SSL Stripping:**
    - Downgrades HTTPS → HTTP to steal info.
3. **Cookie Hijacking (if no Secure flag):**
    - Session cookies stolen.
4. **Expired/Invalid Certificates:**
    - User may ignore warnings and still get phished.

3.6 Making a Request

Example Response:

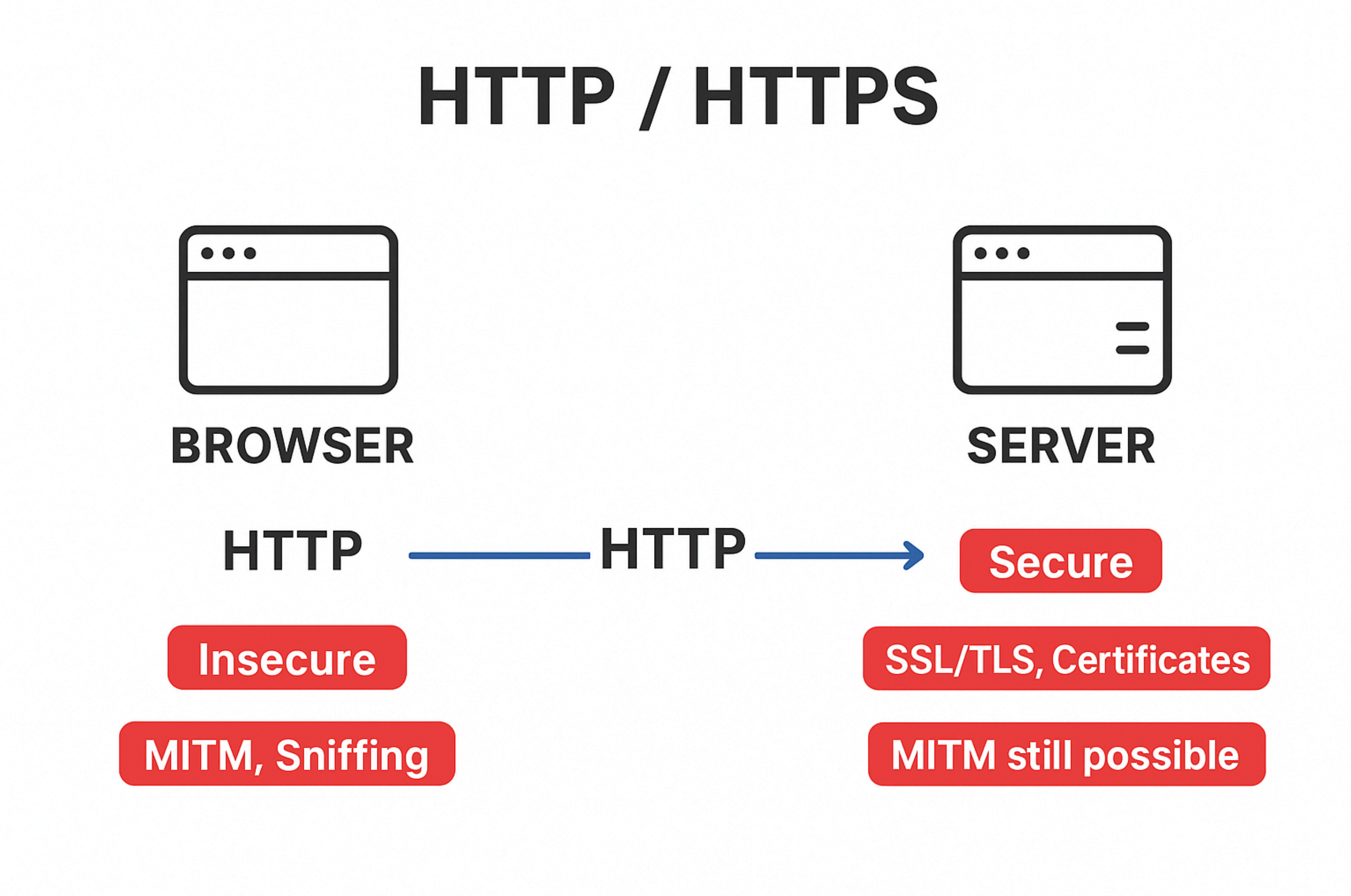
```
HTTP/1.1 200 OK

Server: nginx/1.15.8
Date: Fri, 09 Apr 2021 13:34:03 GMT
Content-Type: text/html
Content-Length: 98


<html>
<head>
    <title>TryHackMe</title>
</head>
<body>
    Welcome To TryHackMe.com
</body>
</html>
```

To breakdown each line of the response:

- **Line 1:** HTTP 1.1 is the version of the HTTP protocol the server is using and then followed by the HTTP Status Code in this case "200 OK" which tells us the request has completed successfully.
- **Line 2:** This tells us the web server software and version number.
- **Line 3:** The current date, time and timezone of the web server.
- **Line 4:** The Content-Type header tells the client what sort of information is going to be sent, such as HTML, images, videos, pdf, XML.
- **Line 5:** Content-Length tells the client how long the response is, this way we can confirm no data is missing.
- **Line 6:** HTTP response contains a blank line to confirm the end of the HTTP response.
- **Lines 7-14:** The information that has been requested, in this instance the homepage.



4. Headers

- Headers are additional bits of data you can send to the web server when making requests.
- Although no headers are strictly required when making a HTTP request, you'll find it difficult to view a website properly.

4.1 Common Request Headers

These are headers that are sent from the client (usually your browser) to the server.

1. **Host:** Some web servers host multiple websites so by providing the host headers you can tell it which one you require, otherwise you'll just receive the default website for the server.
2. **User-Agent:** This is your browser software and version number, telling the web server your browser software helps it format the website properly for your browser and also some elements of HTML, JavaScript and CSS are only available in certain browsers.

3. **Content-Length:** When sending data to a web server such as in a form, the content length tells the web server how much data to expect in the web request. This way the server can ensure it isn't missing any data.
4. **Accept-Encoding:** Tells the web server what types of compression methods the browser supports so the data can be made smaller for transmitting over the internet.
5. **Cookie:** Data sent to the server to help remember your information (see cookies task for more information).

4.2 Common Response Headers

These are the headers that are returned to the client from the server after a request.

1. **Set-Cookie:** Information to store which gets sent back to the web server on each request (see cookies task for more information).
2. **Cache-Control:** How long to store the content of the response in the browser's cache before it requests it again.
3. **Content-Type:** This tells the client what type of data is being returned, i.e., HTML, CSS, JavaScript, Images, PDF, Video, etc. Using the content-type header the browser then knows how to process the data.
4. **Content-Encoding:** What method has been used to compress the data to make it smaller when sending it over the internet.

---

5. What are Cookies?

- **Cookies = small text files** stored in your browser by websites.
- They contain **key-value pairs** (like `sessionid=12345`).
- Purpose: To **remember state** in the otherwise *stateless* HTTP protocol.

👉 Without cookies, every page load would “forget” who you are (you’d need to log in again every time).

5.1 How Cookies Work (Flow)

1. **User Logs In**

- You enter username/password.

2. **Server Generates Session ID**
    - Example: `sessionid=abc123xyz`.
    - This is unique to you.
3. **Server Sends Cookie in Response**
    `HTTP/1.1 200 OK Set-Cookie: sessionid=abc123xyz; HttpOnly; Secure; SameSite=Strict`
4. **Browser Stores Cookie**
    - Saved in local storage.
5. **Browser Sends Cookie Automatically on Next Request**
    `GET /profile HTTP/1.1 Host: example.com Cookie: sessionid=abc123xyz`
6. **Server Validates Session ID**
    - If valid, you stay logged in.

5.2 Types of Cookies

1. **Session Cookies**
    - Temporary, deleted when you close browser.
2. **Persistent Cookies**
    - Stored on disk with an expiry date.
    - Used for “Remember Me” logins.
3. **Secure Cookies**
    - Sent only over HTTPS (not HTTP).
4. **HttpOnly Cookies**
    - Cannot be accessed via JavaScript → prevents XSS stealing.
5. **SameSite Cookies**
    - Restricts sending cookies across sites → prevents CSRF.

5.3 Common Uses

- Authentication (login sessions).
- Preferences (dark mode, language).
- Tracking (ads, analytics).
- Shopping carts (remember items).

5.4 Security Risks with Cookies

1. **Session Hijacking**
    - If attacker steals session cookie → they log in as you.
2. **Cross-Site Scripting (XSS)**
    - Malicious script steals cookies.
3. **Cross-Site Request Forgery (CSRF)**
    - Attacker tricks you into sending requests with valid cookies.
4. **Cookie Theft via MITM**
    - On HTTP (not HTTPS), cookies can be sniffed.

5.5 Protection Mechanisms

- Always use **HTTPS** (Secure flag).
- Use **HttpOnly** to block JS access.
- Use **SameSite=Strict** to prevent CSRF.
- Regenerate session IDs after login.
- Short expiry times for sensitive cookies.

✅ In short:
Cookies = memory of the web.
They make logins and personalization possible, but if stolen, they can let attackers impersonate you.

---

6. What is a Web Server?

- A **web server** is software (and sometimes hardware) that **stores, processes, and delivers websites** to users over the internet.
- When you enter `www.example.com`, your browser talks to a web server that serves you the webpage.

6.1 How a Web Server Works (Flow)

1. **Browser Sends HTTP/HTTPS Request**
   - Example: `GET /index.html HTTP/1.1`
   - Sent to server’s IP (port 80 for HTTP, 443 for HTTPS).
2. **Web Server Software Receives Request**
   - Common software:
     - **Apache** (most used, open-source)
     - **Nginx** (high-performance, lightweight)
     - **IIS** (Microsoft Internet Information Services)
     - **LiteSpeed**, **Tomcat**, etc.
3. **Server Processes Request**
   - If static file (HTML, CSS, image) → serves directly.
   - If dynamic request (login, API call) → forwards to backend (PHP, Python, Node.js, etc.) and possibly database.
4. **Server Prepares Response**
   - Example: `HTTP/1.1 200 OK Content-Type: text/html Content-Length: 1024`
   - Followed by webpage data.
5. **Browser Receives & Renders Page**
   - HTML + CSS + JS executed → webpage shown.

6.2 Types of Content Served

1. **Static Content**
   - Files like HTML, images, CSS, JS.
   - Fast, doesn’t change.
2. **Dynamic Content**
   - Generated on-the-fly (user-specific).
   - Example: Your Facebook feed.

6.3 Key Features of Web Servers

- **Request Handling:** Handles multiple user requests at once.
- **Logging:** Keeps logs of requests (useful for monitoring & forensics).

- **Load Balancing:** Distributes requests across multiple servers.
- **Security Features:** SSL/TLS, authentication, access control.

6.4 Security Concerns in Web Servers

1. **Misconfigurations**
   - Default credentials, directory listing, unnecessary modules.
2. **Unpatched Software**
   - Old versions with known vulnerabilities.
3. **DDoS Attacks**
   - Overloading server with requests.
4. **Injection Attacks**
   - SQL Injection, Command Injection (when server interacts with backend).
5. **Directory Traversal**
   - Accessing restricted files via `../../etc/passwd`.
6. **File Upload Vulnerabilities**
   - Uploading malicious scripts.

6.5 Example: Facebook Login via Web Server

1. Browser → Sends POST request with credentials.
2. Web server (Nginx/Apache) receives it.
3. Forwards request to backend (e.g., PHP/Python app).
4. Backend checks DB → responds “Login Successful”.
5. Web server sends back HTTP 200 with session cookie.

✅ In short:
A **web server = the “waiter” of the internet**.
It takes requests from browsers, talks to the “kitchen” (backend/database), and serves the response back to you.

---

7. IP Addressing & Ports

What is an IP Address?

- **IP (Internet Protocol) Address** = Unique number that identifies a device on the internet/network.
- Think of it as a **house address** on the internet.

Two types:

1. **IPv4** → 32-bit, written as `192.168.1.1` (4 billion possible).
2. **IPv6** → 128-bit, written as `2001:0db8:85a3::8a2e:0370:7334` (much larger space, future-proof).

👉 Websites can have **one IP** or multiple (load balancing/CDN).

What are Ports?

- Ports = **communication channels** on a device.
- Example: If IP = House address → Port = Specific room.
- Each service listens on a specific port.

Port	Protocol	Use
80	HTTP	Web (insecure)
443	HTTPS	Web (secure)
21	FTP	File Transfer
22	SSH	Remote login
25	SMTP	Email sending
53	DNS	Domain lookups
3306	MySQL	Database

👉 **Websites mostly use port 80 (HTTP) and 443 (HTTPS).**

Why This Matters in Cybersecurity?

- **Port Scanning (Nmap, Netcat):** Attackers scan open ports → find services to exploit.
- **Misconfigured Services:** Example → Database (3306) exposed publicly = critical risk.
- **Port Forwarding/Redirection:** Can hide malicious services.

✅ In short:

- IP = **where the server lives**.
- Port = **which service you're knocking on**.
- Together → `192.168.1.10:443` = HTTPS service on that server.

---

8. Client–Server Model

The **Client–Server model** is the backbone of how the web works.

Who is the Client?

- The **client** is usually your browser (Chrome, Firefox, Edge).
- It **requests** data (like typing `www.google.com`).

👉 Client = **Asks the question.**

Who is the Server?

- The **server** is a powerful computer that stores website files, databases, and services.
- It **responds** to client requests.

👉 Server = **Gives the answer.**

How They Talk (Step by Step)

1. **You type a website name** (`www.example.com`).
2. **DNS resolves** it to an IP address.
3. **Client (browser) sends request** to the server's IP + correct port (usually 80/443).
4. **Server receives request** → checks files or database.

5. **Server sends response** (HTML, CSS, JS, images).
6. **Browser renders the page** → you see the website.

Example

You → “Show me Facebook home page.”
Server → “Here’s the HTML + CSS + JS that builds Facebook’s homepage.”
Browser → “Okay, let me display it properly for you.”

Why It Matters in Cybersecurity

- If client request is **malicious** → Server may get hacked (SQL Injection, XSS).
- If server response is **poisoned** → Client may get hacked (phishing, malware).
- Middle attackers may **intercept traffic** (MITM).

✅ In short:

- **Client asks, server answers.**
- Communication uses **requests & responses** (via HTTP/HTTPS).
- Every website interaction is **client–server based**.

---

9. Web Hosting & Servers

When you create a website, you need a place to **store** it and make it available 24/7 → that’s where **web hosting** and **servers** come in.

What is Web Hosting?

Web hosting = renting space on a **server** (a special computer) so your website files (HTML, CSS, JS, images, databases) can be accessed via the internet.

👉 Example: Hosting companies like **AWS, Hostinger, GoDaddy, DigitalOcean**.

Types of Hosting

1. **Shared Hosting** 🏠
   - Many websites share the same server.
   - Cheap, beginner-friendly.
   - Risk: If one site is hacked → others may be affected.
2. **VPS (Virtual Private Server)** 🏢
   - One physical server is divided into **virtual servers**.
   - More control, better performance than shared.
3. **Dedicated Server** 🏰
   - Full physical server for one website.
   - High cost but maximum power & security.
4. **Cloud Hosting** ☁️
   - Websites run on multiple connected servers.
   - Scalable, reliable (used by big companies).
   - Examples: AWS, Google Cloud, Azure.

How a Web Server Works

- A **web server** is software + hardware that **delivers web pages**.
- Popular web server software:
  - **Apache** (open-source, widely used)
  - **Nginx** (fast, lightweight, handles high traffic)
  - **IIS** (Microsoft's server)

👉 Example:

- Browser requests → `http://example.com/index.html`
- Web server checks → finds `index.html`
- Sends it back → Browser displays it

Role of Databases in Hosting

- Websites aren’t just static files; they also store user data.
- Example: When you log in to Facebook → it checks your **username & password** in a **database** (MySQL, PostgreSQL, MongoDB).

Why It Matters in Cybersecurity

- **Server misconfigurations** (open ports, weak permissions) = easy entry for hackers.
- **DDoS Attacks** can overload servers & shut websites down.
- **Outdated server software** = vulnerable to exploits.

✅ In short:

- Hosting = **where websites live**.
- Web server = **servant delivering pages to browsers**.
- Database = **memory of the website**.

---

10. HTML, CSS, JavaScript (Website Building Blocks)

Every website you see is built on these **three core technologies**:

a.) HTML (HyperText Markup Language)

📌 **Role:** Structure of the webpage (the skeleton).

- Defines what elements appear on the page → text, images, links, forms.
- Written with **tags** like `<h1>`, `<p>`, `<img>`.

👉 Example:

```
<h1>Welcome to My Website</h1> <p>This is a paragraph.</p> <img src="logo.png" alt="My Logo">
```

Browser output:

- A big heading → *Welcome to My Website*
- A paragraph → *This is a paragraph.*

- An image → Logo displayed

---

b.) CSS (Cascading Style Sheets)

📌 **Role:** Styling/Design (the skin + clothes).

- Controls **colors, fonts, layouts, animations**.

👉 Example:

`h1 { color: blue; font-size: 40px; text-align: center; }` p { color: gray; font-family: Arial, sans-serif; }`

Now the heading becomes **blue**, large, and centered.

---

c.) JavaScript (JS)

📌 **Role:** Interactivity & Logic (the brain).

- Makes websites **dynamic** → dropdown menus, popups, form validation, API calls.

👉 Example:

```
alert("Hello, welcome to my site!");
```

When page loads → popup appears with message.

Another example: Checking if a login form is empty:

```
function validateForm() { let username = document.getElementById("user").value; if (username === "") { alert("Please enter a username!"); } }
```

How They Work Together

- **HTML** → Adds a login form
- **CSS** → Makes the form look beautiful
- **JavaScript** → Checks if user entered details

👉 Without CSS/JS → Website looks like a plain Word document.
👉 With CSS/JS → Website looks modern and interactive.

Why It Matters in Cybersecurity

- **HTML Injection** → Attacker inserts malicious HTML.
- **CSS Attacks** (rare but possible) → Leaking data with CSS.
- **JavaScript Exploits** → Cross-Site Scripting (XSS), stealing cookies, phishing popups.

11. Web Application vs. Website

Many people think they're the same, but in reality there's a big difference.

What is a Website?

- A **website** is mostly **static** (fixed content).
- Built with **HTML + CSS + some JavaScript**.
- The same content is shown to everyone (unless very simple interactions).

👉 Examples:

- Blog (WordPress)
- Portfolio site
- News site (basic articles)

👉 Think of it like a **digital brochure**.

What is a Web Application?

- A **web application** is **dynamic & interactive**.
- Built with **HTML + CSS + JavaScript + backend (Python, PHP, Node.js, Java, etc.) + Database**.
- Content changes based on **user actions**.
- Users can **log in, interact, upload, download, buy, chat**.

👉 Examples:

- Facebook, Instagram
- Gmail

- Online Banking
- Amazon, Flipkart

👉 Think of it like a **software program** but inside your browser.

Key Differences

Feature	**Website** 🌐	**Web Application** ⚙️
Nature	Static (read-only)	Dynamic (interactive)
Backend	Usually none	Always present
Database	Not needed	Needed
User Login	Rare	Common
Complexity	Simple	Complex
Examples	Blog, Portfolio	Gmail, Banking

Why This Matters in Cybersecurity

- **Websites**: Mostly at risk of **defacement** (attacker changes homepage).
- **Web Applications**:
  - More features = More attack surfaces
  - Common targets: **SQL Injection, XSS, CSRF, Broken Authentication**
  - Attackers aim to steal **data, accounts, money**.

✅ In short:

- Website = **Static content, informational**.
- Web App = **Interactive, user-based, connected to databases**.

---

12. Web Security Basics

Security is what protects websites & web apps from hackers.
If we don’t secure them → attackers can steal data, deface sites, or take full control.

a.) HTTPS (Secure Communication)

- **HTTP** = data sent in **plain text** (hackers can sniff passwords).
- **HTTPS** = uses **SSL/TLS encryption** → protects data from eavesdropping.
- Websites should **always** use HTTPS.
  👉 Example: Online banking must be HTTPS or attackers can steal credentials.

---

b.) Firewalls

- A **firewall** filters incoming & outgoing traffic.
- **Network Firewall** = Protects servers from bad traffic.
- **Web Application Firewall (WAF)** = Protects against common web attacks (XSS, SQL Injection).
  👉 Example: Cloudflare, AWS WAF.

---

c.) Authentication & Authorization

- **Authentication** = Who are you? (Login with username/password, OTP, biometrics).
- **Authorization** = What can you do? (Admin vs Normal User).
  👉 Example: You log in → Authentication. You can’t access admin panel → Authorization.

---

d.) OWASP Top 10 (Most Common Web Vulnerabilities)

These are the **biggest threats** to web apps (must-know for cybersecurity):

1. **Broken Access Control** → Users accessing admin data.
2. **Cryptographic Failures** → Weak encryption (password leaks).
3. **Injection Attacks (SQL, NoSQL, OS)** → Attacker runs malicious code in DB.
4. **Insecure Design** → Bad architecture (security ignored).
5. **Security Misconfigurations** → Exposed admin panels, default passwords.
6. **Vulnerable Components** → Using outdated libraries/plugins.
7. **Identification & Authentication Failures** → Weak login systems.

8. **Software & Data Integrity Failures** → Supply chain attacks.
9. **Security Logging & Monitoring Failures** → Breach goes undetected.
10. **Server-Side Request Forgery (SSRF)** → Server tricked into fetching internal data.

Common Security Practices

✅ Always use HTTPS
✅ Keep servers & software updated
✅ Strong authentication (MFA)
✅ Sanitize user input (prevent injection/XSS)
✅ Regular security testing (Pentesting, Bug Bounty)
✅ Backup data & logs

Why It Matters

- Companies lose **millions** in data breaches.
- Security = Trust. Without it, even big websites fail.

✅ In short:

- **HTTPS protects communication.**
- **Firewalls block attacks.**
- **Auth ensures identity & permissions.**
- **OWASP Top 10 = Web hacker’s favorite playground.**

---

14. TCP/IP and OSI Model Notes

TCP/IP Model

Definition:

- **TCP/IP (Transmission Control Protocol / Internet Protocol)** is the **protocol suite** that

governs communication over the Internet.
- It defines how data is **packaged, addressed, transmitted, routed, and received**.

Layers of TCP/IP Model:

Layer	Function	Protocols / Examples
Application Layer	Provides network services to applications	HTTP, HTTPS, FTP, SMTP, DNS
Transport Layer	Ensures end-to-end communication, error detection, and flow control	TCP (reliable), UDP (unreliable)
Internet Layer	Handles logical addressing and routing	IP, ICMP, ARP
Network Access / Link Layer	Manages physical transmission and access to network hardware	Ethernet, Wi-Fi, MAC addresses

Key Points:

- **TCP**: Reliable, connection-oriented communication.
- **UDP**: Faster, connectionless, no guarantee of delivery.
- TCP/IP is **practical and widely used** in real-world networks.

OSI Model

Definition:

- **OSI (Open Systems Interconnection) Model** is a **conceptual framework** used to understand network interactions.
- It has **7 layers**, each with specific functions.

Layers of OSI Model:

Layer	Function	Examples
7. Application	Interface for user applications	HTTP, FTP, SMTP
6. Presentation	Data translation, encryption, compression	SSL/TLS, JPEG, ASCII
5. Session	Manages sessions between applications	NetBIOS, RPC

Layer	Function	Examples
4. Transport	Reliable data transfer, segmentation, error detection	TCP, UDP
3. Network	Logical addressing and routing	IP, ICMP
2. Data Link	Frames, MAC addressing, error detection	Ethernet, Wi-Fi
1. Physical	Transmission of raw bits over physical medium	Cables, Switches, Hubs

Key Points:

- OSI is **theoretical**, helps understand networking concepts.
- TCP/IP is **practical**, used in real-world networking.
- OSI layers 5-7 are sometimes grouped in TCP/IP **Application Layer**.

```
                        END of WEB BASICS
```