***Common Application Layer Protocols***

*Author: Vineeth M*

*Date: sep 23 , 2025*

***HTTP (Hypertext Transfer Protocol)***

***HTTP (Hypertext Transfer Protocol)*** *is a communication protocol that allows web browsers and web servers to exchange information. When you enter a website’s URL, your browser sends an HTTP request to the server, and the server responds with the necessary data to display the webpage. Think of it as a common language that ensures both the browser and the server understand each other so the page loads correctly.*

*When you open a browser and type a website URL, the process begins with a* ***DNS lookup*** *to find the IP address of the website—similar to looking up a phone number. Once the IP address is identified, the browser sends an* ***HTTP request*** *to the server, asking for the webpage resources. The server processes this request and sends back an* ***HTTP response*** *containing files such as HTML, CSS, and images. Finally, the browser interprets this data, assembles it, and renders the webpage on your screen, ready for you to view and interact with.*

### ***Usage Scenarios of HTTP***

1. ***Browsing Websites***
   * *When we visit any website your browser uses HTTP to request web pages and receive content like text, images, and videos.*
2. ***Submitting Forms***
   * *When you fill in a sign-up form or search for something online, the browser sends your input as an HTTP request to the server, which processes it and returns results.*
3. ***Downloading Files***
   * *HTTP allows you to download files such as PDFs, images, or software from websites by transferring them from the server to your computer.*
4. ***APIs and Web Services***
   * *Many apps and services use HTTP to communicate with servers. For example, a weather app fetches live weather updates through HTTP-based API requests.*
5. ***Embedded Content***
   * *HTTP is used to load additional resources (like ads, videos, or widgets) from other servers into a webpage.*

# *HTTPS (HyperText Transfer Protocol Secure)*

***HTTPS (HyperText Transfer Protocol Secure)*** *is the secure version of HTTP and is the most widely used protocol for data transfer between a web browser and a website. Unlike regular HTTP, HTTPS encrypts the communication, ensuring that sensitive information like login details, payment data, or personal information is protected from attackers. Modern browsers, such as Google Chrome, highlight secure websites with a padlock icon in the URL bar, while non-HTTPS sites are marked as “Not Secure,” making HTTPS essential for any website that handles user data.*

*HTTPS works by combining the standard HTTP protocol with* ***SSL (Secure Socket Layer)*** *or its updated version,* ***TLS (Transport Layer Security)****. While the workflow of requesting and responding to data remains the same as in HTTP, the added SSL/TLS layer provides encryption, authentication, and data integrity. This means all information exchanged between the browser and the server is encrypted, unreadable to third parties, and securely delivered, ensuring both safety and trust in web communication.*

***Why HTTPS Matters***

*HTTPS is essential because it secures communication between browsers and servers, protecting data from being intercepted or stolen. With regular HTTP, information is sent in plain text packets that can be easily captured using simple tools, making users especially vulnerable when browsing on public Wi-Fi. Without encryption, sensitive details such as passwords, credit card numbers, or personal information can be exposed to attackers. HTTPS solves this problem by encrypting the data so that even if packets are intercepted, they appear as unreadable strings of random characters instead of meaningful text.*

***How HTTPS Provides Security***

*HTTPS achieves this protection by using SSL/TLS encryption and digital certificates. SSL ensures that the browser communicates directly with the intended server and that only the communicating systems can access the exchanged data. It relies on a* ***public-private key pair****: the public key encrypts the data, while the private key (kept securely on the server) decrypts it. This ensures confidentiality, integrity, and trust in communication. Beyond security, HTTPS also improves performance and user confidence, as modern browsers flag non-HTTPS sites as “Not Secure.” The key advantages of HTTPS include secure communication, data integrity, user privacy, and, in many cases, faster transmission speeds.*

## ***HTTP vs HTTPS***

|  |  |
| --- | --- |
| **HTTP** | **HTTPS** |
| Data is sent in **plain text**, vulnerable to interception | Data is **encrypted** using SSL/TLS, secure from eavesdropping |
| Suitable for **non-sensitive content** (blogs, simple websites) | Essential for **sensitive content** (banking, login pages, e-commerce) |
| HTTP Works at the Application Layer | Operates at the **Application Layer** but adds security at the **Transport Layer** |
| No SSL/TLS certificate required. | Requires an **SSL/TLS certificate** for verification |

***HTTPS Attacks***

***SSL/TLS stripping***

***SSL/TLS stripping*** *is a type of attack where an attacker tries to make your secure HTTPS connection become an unencrypted HTTP connection. When you type a website address, your browser usually starts with HTTP and then is redirected to HTTPS. A hacker sitting between you and the website can intercept this process and change the redirect so your browser stays on HTTP. This means all the information you send, like passwords or messages, can be* ***seen or stolen*** *by the attacker. Websites can protect users from this attack by using* ***HSTS (*HTTP Strict Transport Security*)****, which forces browsers to always use HTTPS.*

***Man-in-the-Middle (MITM) Attacks***

*In HTTPS, a* ***man-in-the-middle (MITM)*** *attack happens when someone secretly positions themselves between a user and a website to intercept or change data. However, HTTPS normally prevents this because all data is encrypted and verified using a* ***digital certificate****. The attacker can only read or alter traffic if they somehow trick the browser into accepting a* ***fake certificate*** *or force the connection to downgrade to insecure HTTP. When the browser correctly checks the certificate’s validity and signature, such attacks fail, keeping communication private and authentic between the user and the real website.*

***How HTTPS SSL/TLS Certificates and Certificate Authorities Prevent Man-in-the-Middle Attacks***

*In HTTPS, a* ***certificate*** *acts like an ID card for a website, proving its identity to your browser. It contains the website’s name, its public key for encryption, and a digital signature from a trusted* ***Certificate Authority (CA)****. The CA is like an official organization that verifies a website’s identity and issues certificates only to legitimate owners. When you visit a website, your browser checks if the certificate is valid, correctly signed by a trusted CA, and matches the site’s name. If everything is correct, the connection is encrypted and secure. However, in a* ***man-in-the-middle attack****, an attacker tries to intercept or change data between you and the website. This usually fails in HTTPS because the attacker’s fake certificate won’t be trusted by your browser, which warns you that the connection is not secure.*

***Certificate Spoofing***

*Certificate spoofing is when an attacker creates or uses a fake SSL/TLS certificate to pretend to be a real website, tricking a browser or user into thinking the connection is secure so the attacker can read or alter data. Browsers normally block these fakes, so spoofing works only if the attacker makes the browser trust the fake (for example by installing a rogue root certificate, stealing a real private key, or using a compromised CA). To stay safe, never ignore certificate warnings and avoid installing unknown root certificates.*

***Session Hijacking***

*Session hijacking is when an attacker steals the temporary key (a session cookie or token) that proves you’re logged into a website and then uses it to act as you without needing your password. This can happen if the token is grabbed over an insecure Wi‑Fi network, taken by malicious software on your device, or stolen by a script from a vulnerable website (XSS). HTTPS helps protect tokens while they travel, but to stay safe use HTTPS, enable 2FA, log out on shared devices, and avoid untrusted networks or suspicious software.*

***Mixed Content Attacks***

***mixed content attacks*** *happen when a website is mostly secure (HTTPS) but still loads some parts like images, scripts, or videos over* ***insecure HTTP****. Because those parts aren’t encrypted, an attacker on the same network can* ***change them or add malicious code****, which can steal data or make the page do bad things.*

***DNS Protocol***

*The* ***Domain Name System (DNS) protocol*** *is a system that allows users to access websites using easy-to-remember hostnames instead of numeric IP addresses. It acts like the internet’s phonebook, translating domain names such as* [*www.example.com*](http://www.example.com/) into their corresponding IP addresses (IPv4 or IPv6) so computers can communicate. DNS relies on structured records, called **resource records**, stored on **authoritative name servers**, and includes information like hostnames, IP addresses, subdomains, and caching times (TTL). By following a hierarchical structure of domains and zones, DNS ensures that every website has a unique name and can be efficiently located by users.

*When a user types a domain name into a browser, the DNS protocol initiates a query to resolve the name into an IP address. The browser first checks its cache, then the OS’s stub resolver, and if unresolved, the request is sent to a* ***recursive DNS server*** *(often provided by an ISP). If needed, the recursive resolver queries higher-level servers: the* ***root server****, the* ***TLD server****, and finally the* ***authoritative name server****, which contains the definitive IP address. The response is then returned to the browser, allowing it to connect to the web server.* DNS primarily uses **UDP** for most queries because it is fast and efficient, while **TCP** is used for larger responses or tasks like zone transfers that require reliability. This combination allows DNS to quickly and accurately translate domain names into IP addresses, supporting everything from everyday web browsing to advanced enterprise systems with features like load balancing, geographic routing, and multi-CDN configurations.

# File Transfer Protocol (FTP)

**File Transfer Protocol (FTP)** is one of the earliest and most widely used ways to transfer files on the internet. It belongs to the application layer of the OSI model and is designed to move files between a client (such as a user’s computer) and a server (a remote system). FTP is reliable because it can transfer files even if the systems involved use different operating systems or file structures. Unlike HTTP, which is mainly used for web pages, FTP focuses only on transferring files like text, images, and programs, making it a dedicated and efficient tool.

FTP is a standard communication protocol that hides the technical differences between systems, such as file structures or character sets, so the transfer works smoothly. It supports different file formats, including text formats like ASCII, EBCDIC, and binary formats for images or program files. ASCII is often the default for text, while binary mode is used for other types of files. Because of its ability to handle different types of data and ensure reliable transfers, FTP is still commonly used today for uploading, downloading, and managing files on servers across the internet.

**Types of FTP**

* **Anonymous FTP**: Some websites let anyone download files without logging in. You just use “anonymous” as the username and “guest” as the password. But access is limited—you can usually only download files, not explore folders.
* **Password-Protected FTP**: This type needs a username and password to connect. It gives better control and security than anonymous FTP.
* **FTP Secure (FTPS)**: This is a safer version of FTP. It uses encryption (TLS) so that the files being transferred are protected.
* **FTP over Explicit SSL/TLS (FTPES)**: This starts as a normal FTP connection on port 21, then switches to an encrypted, secure connection.
* **Secure FTP (SFTP)**: Even though it sounds like FTP, it’s different. SFTP uses the SSH protocol (port 22) and is very secure, which is why many organizations prefer it.

****How FTP Works****

*FTP (File Transfer Protocol) is a* ***client-server protocol*** *used to transfer files between a user’s computer (client) and a remote server. It uses* ***two separate channels****: a* ***control channel*** *for sending commands and managing the session, and a* ***data channel*** *for transferring the actual files. To start, a user usually logs in to the FTP server, though some servers allow public access without a login, called* ***anonymous FTP****. After connecting, the client can perform actions like downloading, uploading, deleting, or renaming files on the server.*

*FTP maintains a session with the server, meaning it keeps track of the user throughout the connection, unlike HTTP, which is stateless. The* ***control connection*** *(usually on port 21) sends login credentials and commands, while the* ***data connection*** *(usually on port 20) transfers the files themselves. The client software on the user’s computer handles both connections, sending requests and receiving files from the server. FTP can also work in* ***active or passive modes****, which determine how the server and client establish the data connection. By keeping control and data separate, FTP ensures reliable and organized file transfers between systems.*